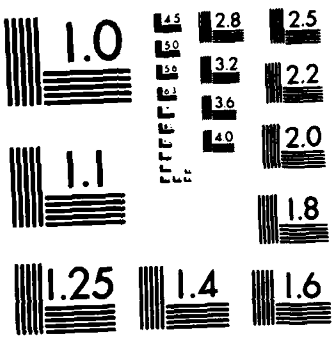
AD-A143 753 THE BATTLEFIELD ENVIRONMENT MODEL (BEM)(U) MITRE CORP 1/3
MCLEAN VA MITRE C3I DIV R S CONKER ET AL. SEP 83
MTR-83W00245 F19628-84-C-0001

UNCLASSIFIED F/G 15/7 NL



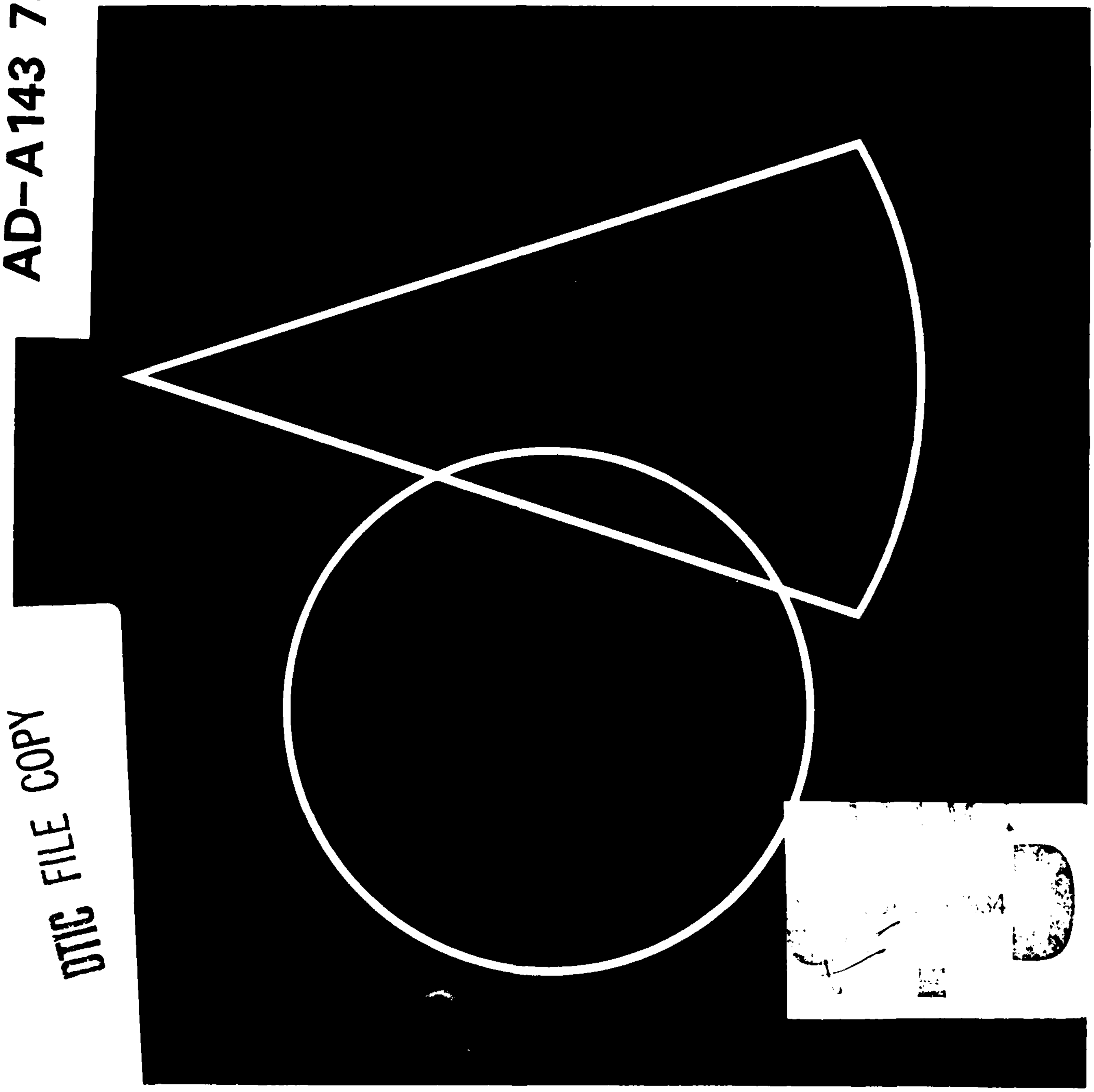
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

2

The Battlefield Environment Model (BEM)

AD-A143 753

MTR-83W00245



DTIC FILE COPY

MITRE

The Battlefield Environment Model (BEM)

R. S. Conker
J. R. Davidson
P. K. Groveston
R. O. Nugent

September 1983

MTR-83W00245

SPONSOR:
Army Model Management Office
CONTRACT NO.:
F19628-84-C-0001

~~This document was prepared for authorized distribution.~~
~~It has not been approved for public release.~~

The MITRE Corporation
MITRE C^{3}I Division
Washington C^{3}I Operations
1820 Dolley Madison Boulevard
McLean, Virginia 22102

MITRE Department
and Project Approval:

ABSTRACT

This report describes the Battlefield Environment Model (BEM), a computer model implemented by MITRE for use in conducting simulations of military combat. Specifically, the model simulates Red unit observables against which Blue sensors can operate. The modeling effort employs object-oriented programming techniques using the RAND-developed ROSS programming language. The BEM provides a stream of realistic sensor reports for ANALYST, an expert system for determining battlefield activity. In addition, the BEM provides an experimental environment for studying C[2] structures and various approaches to sensor tasking. Future enhancements of the BEM are also presented.

Command and control system

TABLE OF CONTENTS

TABLE OF CONTENTS
(Concluded)

LIST OF ILLUSTRATIONS

LIST OF TABLES

1.0 INTRODUCTION

1.1 Background and Purpose

This report was prepared to provide a description of the MITRE Battlefield Environment Model (BEM), a computer model used in conducting simulations of military combat. The BEM model is designed to generate Red unit observables against which Blue sensors can operate; specifically, it is designed to generate sensor observables from a Soviet division-sized force at company resolution. With this model, enemy tactical events such as movement, communications, artillery and air defense firings, and radar emissions are generated and subsequently detected by Blue (NATO) force sensors. The resulting stream of sensor reports is sent by the BEM to the ANALYST, an expert system used to determine enemy tactical disposition on the battlefield.

Both the BEM and the ANALYST are enhancements of previous models which employed conventional, procedural programming techniques. The predecessor of the BEM was the MITRE Threat Event Generator (MTEG)[1,2], a model written in Pascal which generated sensor reports by parametrically filtering the observables. The BEM evolved from this model through three significant enhancements:

- a more realistic threat representation.
- a cause and effect modeling of sensor detections.
- implementation using object-oriented techniques and the ROSS programming language (to be discussed).

The current ANALYST[3] is LISP-based and is operational on both a VAX 11/780 and a LISP machine.

1.2 Object-Oriented Programming

1.2.1 General

The essence of object-oriented programming is that it is centered around objects (actors in the simulation) which communicate with each other by way of messages. Features of the technique provide promise to overcome many of the limitations of conventional procedural programming in the areas of intelligibility, modifiability, understanding, and performance characteristics. (See MITRE WP83-W004074 for an evaluation of the use of object-oriented programming).

The remainder of this section provides a discussion of object-oriented languages in general and of ROSS, the object-oriented language in which the BEM is implemented.

1.2.2 Programming Using Object-Oriented Languages

The use of object-oriented programming for simulation purposes requires that simulation objects be created to represent the systems being simulated (e.g., the military units). Those objects must have characteristics and behaviors sufficiently defined for them to adequately portray the system or units being represented.

Objects have property lists and behaviors associated with them. The properties can represent such items as the numbers of vehicles, location, and radio frequencies. Behaviors are the actions taken by the objects upon receiving a message. Both properties and behaviors may be modified during the simulation.

There are two types of objects called basic and auxiliary objects. Basic objects are actors in the simulation and have some real-world representation such as the military units being simulated. Auxiliary objects assist in the construction and conduct of the simulation; an example is the Mathematician, which centralizes most of the computation, thus making the basic object behaviors cleaner and therefore more readable. Basic objects may be generic objects, to represent classes of objects, or instance objects, to represent the actual unit or object being simulated.

Objects are organized into a hierarchy which permits them to inherit properties and behaviors from ancestor objects in the hierarchy. This allows the programmer to establish the property or behavior at the highest common level and aids in the construction of the simulation. Inherited properties may be modified at the lower level before or during the simulation. An example of a hierarchy used for implementation is depicted in Figure 1-1. The top-level object of the hierarchy is called "Something". "Something" has properties and behaviors which allow it to be called upon to create offspring. In turn, these offspring at the generic level can be called upon to create their own offspring at either the generic or the instance level. The example shows how a generic tank company might be created and how it in turn could be asked to create an instance tank company, one of the basic objects in the simulation.

All action in the simulation is controlled through message passing and the object behavorial rules. Object behaviors are in the form of IF-THEN rules which list the actions to be taken by an object on receipt of particular messages. These actions frequently involve sending messages to other objects or planning for action to be taken at a future time in the simulation.

1.2.3 ROSS

The particular object-oriented simulation system chosen to implement the BEM was the RAND Corporation's Rule Oriented Simulation System (ROSS). ROSS was developed by RAND specifically to demonstrate the application of artificial intelligence techniques to military combat simulation.[5,6]

1.2.3.1 ROSS Commands. ROSS is invoked through the use of commands in the ROSS language. The format of a ROSS command is

(< ask or tell > < object > < message >)

where the parentheses set off a list as in LISP, the language in which ROSS is based, and the greater-than/less-than (> , <) signs set off parts of the command for explanation purposes. "Ask," used interchangeably with "tell," alerts the object to receive a message, that is, it involves a LISP macro to process the remainder of the list, sending the message to the object.

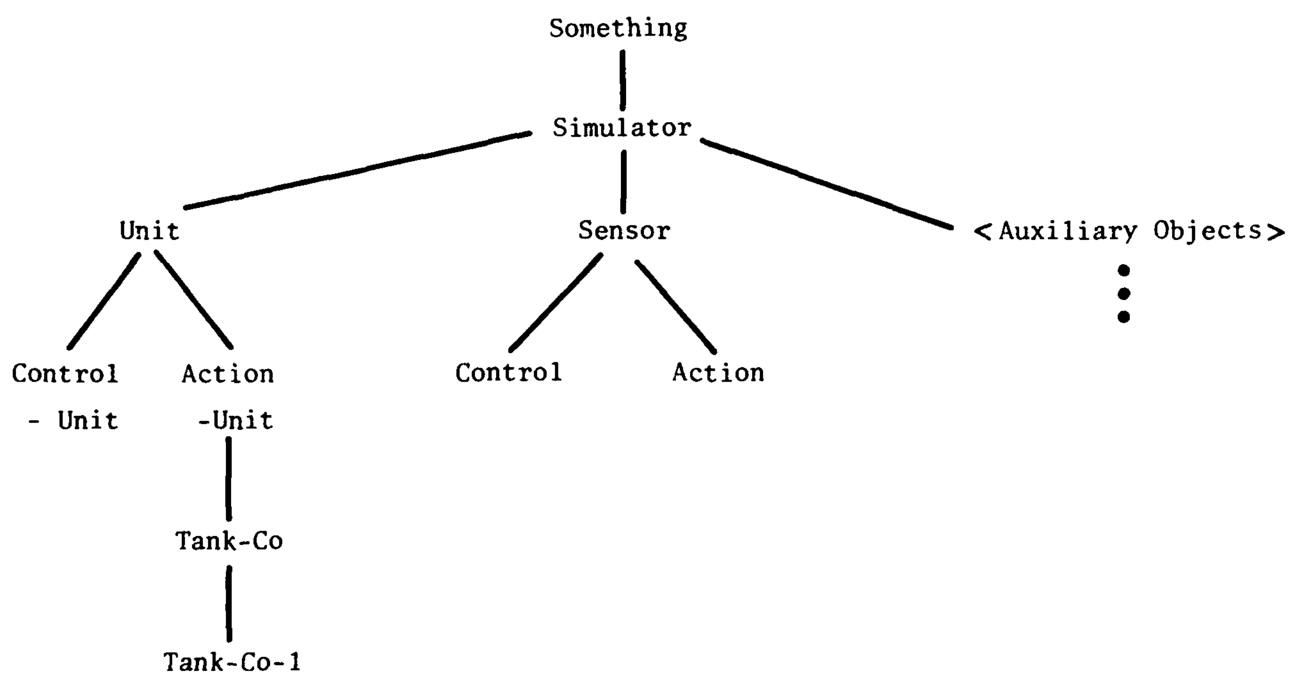


FIGURE 1-1
OBJECT HIERARCHY

1.2.3.2 ROSS Behaviors. The set of actions which an object takes receipt of a message constitutes what is known as a behavior. The command to give an object a behavior is of the form

(<ask or tell> <object>
when receiving
<message>
< action(s)>)

Figure 1-2 gives an example of how a previously created generic "Action-Unit" could be tasked to create another generic unit "Tank Action-Unit would inherit the behavior ability to create another object the top-level object Something. The object property list is in the fo slot-value pairs, e.g., "tanks", and "10," indicating that each tank cor begins with ten tanks. The semi-colons set off comments and are us make the code more readable or to provide explanation.

Figure 1-3 gives an example of how a generic unit could be task create instances of itself and provide additional properties to the instanc

An example is given in Figure 1-4 of an object behavior. The ex also demonstrates some of the features of the ROSS language, in part the abbreviation package which aids in the readability of the code. behavior is given to the generic object "Unit" and would therefore be inh by all descendants, both generic and instantiated, of Unit. This bel consists of four actions which include sending messages to other a sending messages to the unit itself, and modifying unit properties.

Another feature of the ROSS language demonstrated in the exam that of pattern-matching. This feature allows the use of symbolic variat the messages (preceded by > or + symbols in the message and evalua the action part of the behavior when preceded by ! or &).

1.3 Organization of the Report

Section 2.0 provides an overview of the BEM model, a descriptio BEM simulation, and a description of the components of the BEM/ANAL

```
(ask action-unit create generic tank-co

          with tanks      10     ;
          day-length      0.5    ;  Kms
          night-length    0.3    ;  Kms
          day-speed       20.0   ;  Kmph
          night-speed     15.0   ;  Kmph
          time-to-clear   5      ;  minutes
          time-to-close   5      ;  minutes
          fuel-cons       280    ;  tons per day
          ammo-cons       50     ;  tons per day
          shape           8      ;  graphics font
)
```

FIGURE 1-2
EXAMPLE GENERIC OBJECT CREATION

```
(ask tank-co create instance tank-co-0355
    with beg-end-loc  (51.4 30.8)  (50.9 30.4)
              superior              tank-bn-cop-55
              start-time            291.1
              stop-time             344.9
              plan                 ((296.6 node 91)
                                     •
                                     •
                                     •
                                    (329.6 node 136))
              status               moving
              cmd-up-freq          43.3
              position             (51.4 30.8)
              activity             assembled
```

FIGURE 1-3
EXAMPLE INSTANCE OBJECT CREATION

```
(ask unit when receiving (turn >type radar >onoff)
(tell sector ! myself has turned ~the type radar ! onoff)
(tell artist ! myself has turned ~the type radar ! onoff)
   at ! (~your position))
(tell ! (~your superior) ! myself has turned ~the type
   radar ! onoff)
```

Legend:

- `!` evaluate this expression (as ROSS is a non-evaluating dialect of LISP)
- `>` indicates a variable
- `~` indicates an abbreviation
  e.g. "~the" also means evaluate this expression

FIGURE 1-4
EXAMPLE OBJECT BEHAVIOR DEFINITION

work stations. A more detailed description of the BEM is presented in Section 3.0, which details the "actor files" in the BEM simulation. Section 4.0 describes future enhancements of the BEM. Appendix I and II contain sensor and threat research for the BEM. Data lists of threat representation for the BEM are presented in Appendix III, which is classified SECRET and published separately. Appendix IV presents the ROSS-Language code for the BEM, while Appendix V describes the BEM development and demonstration facilities.

2.0 GENERAL DESCRIPTION OF THE BATTLEFIELD ENVIRONMENT MODEL (BEM)

2.1 Model Overview

As noted in Section 1.0, the primary purpose of the BEM is to produce realistic sensor reports for the ANALYST, MITRE's expert system for intelligence fusion. The BEM also provides a testbed environment where various C^2 structures and approaches to sensor tasking can be modeled. Figure 2-1 shows a high-level view of the BEM/ANALYST models.

In addition to the computer code necessary to run a BEM simulation, there is a database related to the movement of a Soviet division-level force in the Fulda Gap of Germany. This data originates in the Scenario Oriented Recurring Evaluation System (SCORES)[7] which provides battalion-level resolution. Previous work at MITRE[8,9] has expanded this data to company level, and from this, beginning and ending times and positions have been generated (See Section 2.2.1). Path construction routines which can be used dynamically during a simulation are also used prior to a BEM run to generate routes for all threat units. These routes are constructed with the aid of a nodal network which includes the road network and any other trafficable path for a company-sized unit.

Currently, sensor detection in the BEM extends only to the point of determining whether a given observable event has taken place within the geographic coverage area of a relevant sensor. Other considerations related to sensor detection, such as terrain masking and signal attenuation, will reside in formal sensor models and appear in a future version of the BEM.

2.2 Input Data

2.2.1 Red Units

Red units in the BEM represent a first-echelon Soviet division in the Fulda Gap area of Germany on D + 1, i.e., one day after the start of hostilities. The division consists of three tank regiments, one motorized rifle

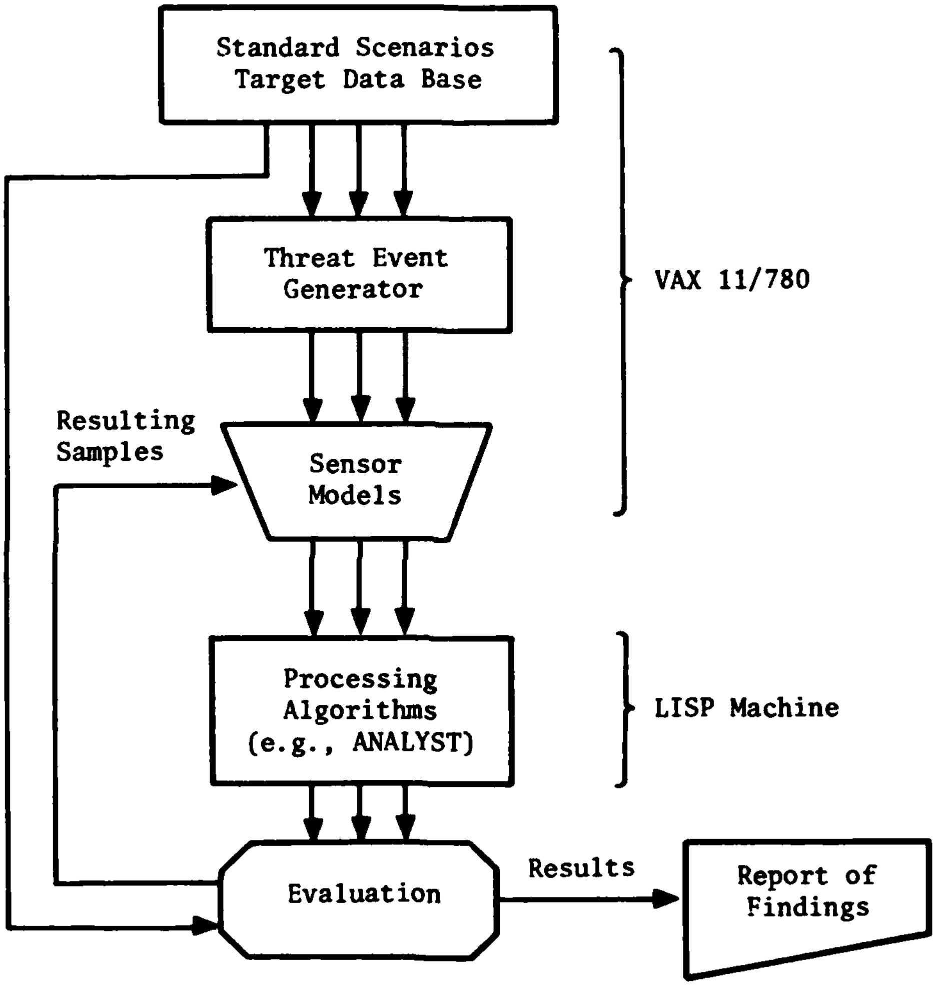


FIGURE 2-1
HIGH LEVEL VIEW OF THE BEM/ANALYST MODELS

regiment, their regimental artillery groups, an artillery regiment, an air defense battalion, and combat service support units. Units are represented down to company level, with headquarters represented from battalion to division level. Unit positions are developed for 0330 hours and 0930 hours on D + 1 of SCORES European Sequence II a. Figure 2-2 depicts the command structure.

Common unit data are developed and stored at the generic unit level. These data include the number and types of vehicles and radios and other characteristics such as march length and speed and radio nets. Individual unit data such as location and radio frequency are developed and stored at the instance unit level. Appendix III (published separately) presents representative data at both the generic and instance levels.

2.2.2 Blue Sensors

The BEM employs five generic types of sensors:

- MTI (moving target indicators) which detect unit movement on the battlefield.
- COMINT (communications intelligence) which detects radio communications.
- CMCB (counter-mortar/counter-battery) which detects mortar, rocket, missile and artillery firings.
- ELINT (electronic intelligence) which detects radar emissions.
- IMINT (imagery intelligence, specifically photo).

CMCB is only represented on the ground while IMINT is only represented airborne. All others are represented both on the ground and airborne. COMINT sensors are omnidirectional. All others are directional, with IMINT having a squared coverage area directly beneath the platform, and MTI, CMCB and ELINT having a fan-shaped coverage.

2.3 Description of a BEM Simulation

The BEM has a scenario file with particular events set to occur at specified times during the simulation. First among these events is the

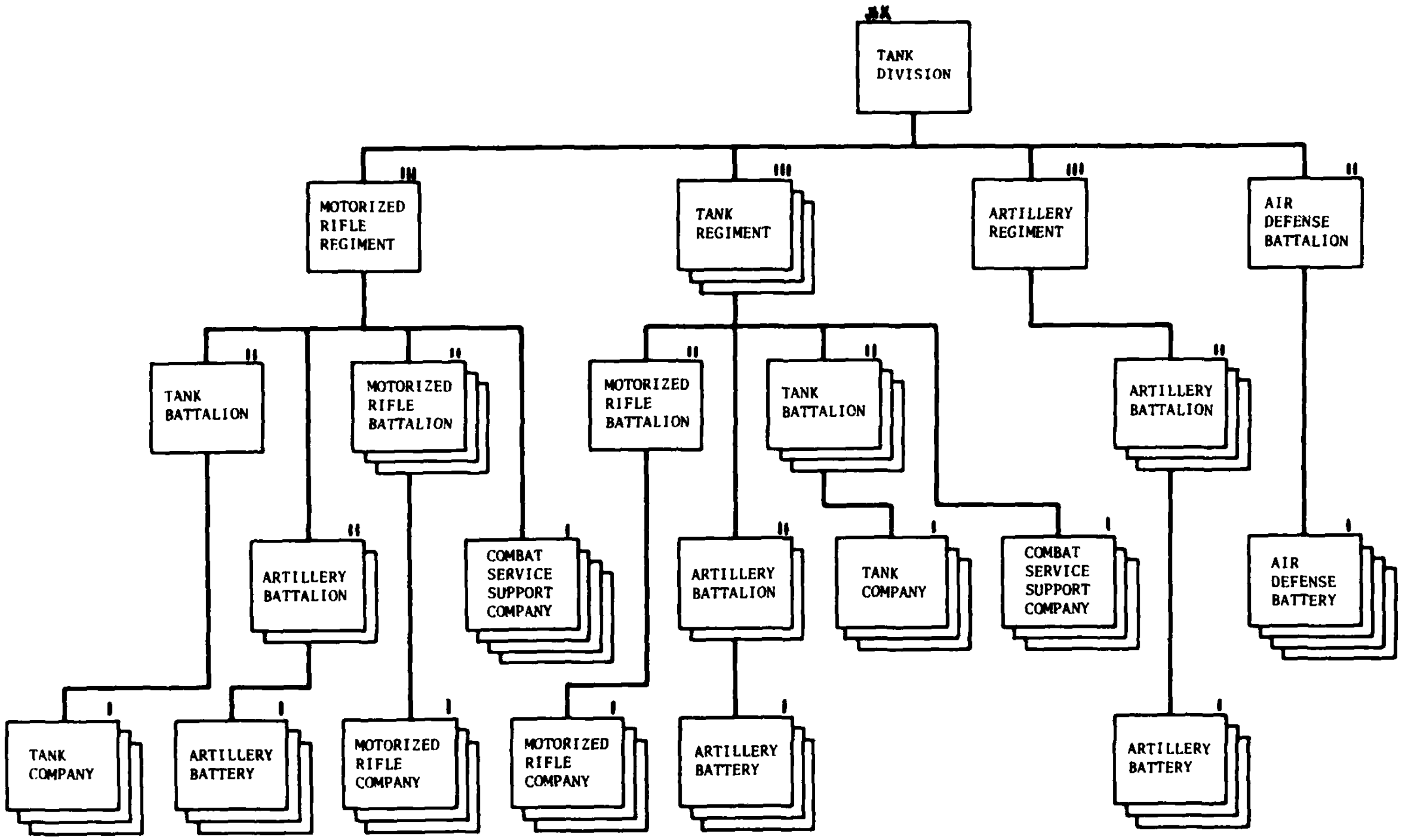


FIGURE 2-2
RED UNITS IN THE BEM

delivery of orders down through the Soviet military chain of command beginning at division level and proceeding to the company/battery level. Movement of these threat units will not begin before their start times as prescribed by the SCORES data.

A second major event set in motion by the scenario file is the initiation of artillery firings, and anti-aircraft radar and firing activity. In a two-sided simulation, many of these events would be generated on the Red side in response to events occurring on the Blue side. Since the current BEM is one-sided, it is necessary to generate these events artificially. Once set in motion, these activities can continually be produced throughout the simulation by Monte Carlo techniques.

The third major activity initiated by the scenario file is the tasking of ground and airborne sensors. Ground sensors will travel along the node network until they reach a point where they will stop and begin their mission. Passive sensors (COMINT, ELINT) will remain on for the duration of their mission, while active sensors (MTI, CMCB) will periodically turn on and off. Airborne sensors will start from and return to a specified airport. Airborne MTI, ELINT and COMINT sensors will fly to designated points and proceed to traverse racetrack patterns with their sensors turned on during each leg of the racetrack and off during each turn. Photo-imagery sensors will be given specified flight points over which a picture will be taken. Although this preprogramming of sensor tasking allows the simulation to run systemically, it is nevertheless possible to interactively task sensors during the running of the simulation.

As previously noted, the current BEM employs object-oriented programming techniques using the ROSS programming language. Events and calculations occur in response to messages sent amongst the various objects. Part of the BEM Controller Station is a text terminal which, in addition to allowing interaction with the BEM, can display the ROSS messages sent during a simulation. An example of some of these messages is shown in Figure 2-3. The format of a line of the message is the time the message was sent, the

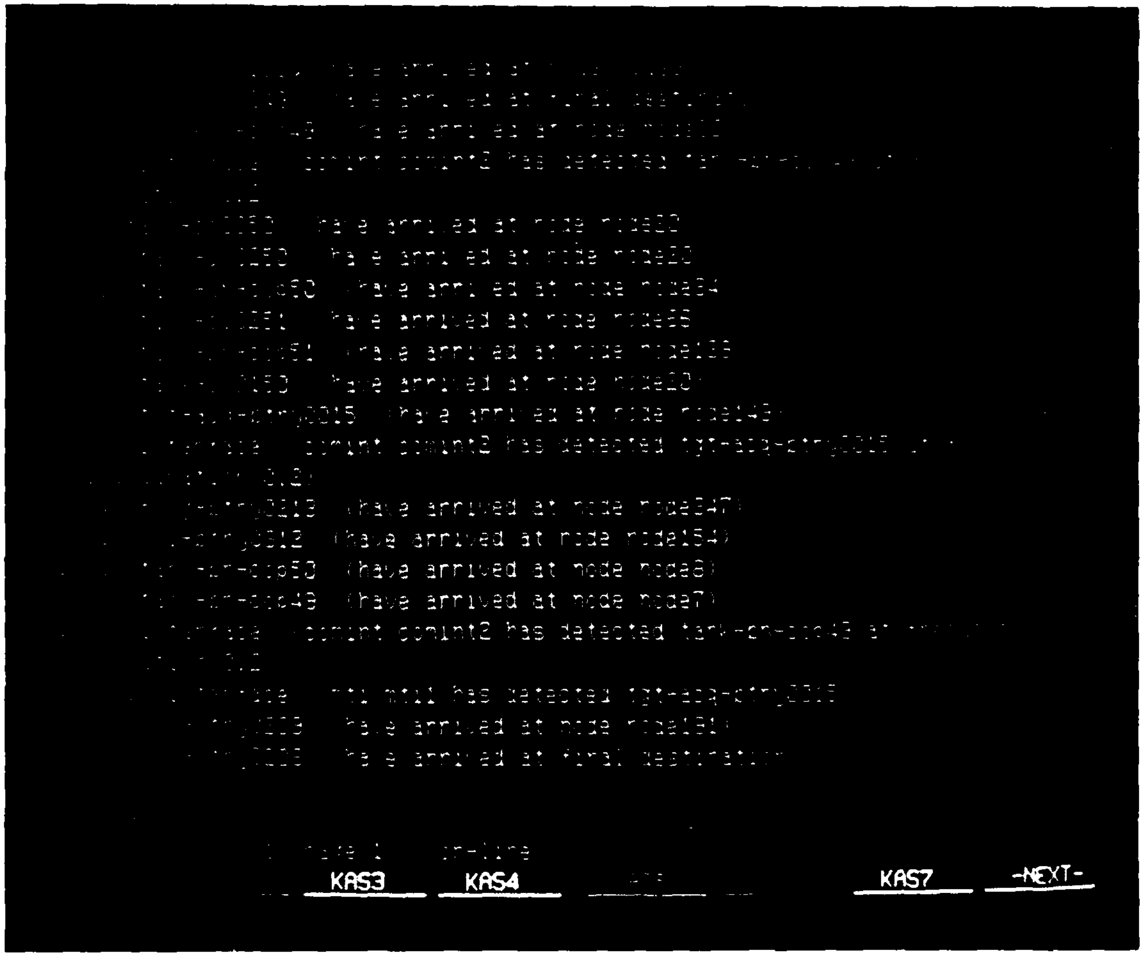


FIGURE 2-3
EXAMPLE OF ROSS MESSAGES ON THE TEXT TERMINAL

object to whom the message was sent, and the message itself with any variables evaluated (as is) in it. The user may have all messages print out or only those of particular interest. The messages may also be sent to a file on the disk. The complete set of messages is an invaluable aid to debugging, whereas a carefully chosen subset is displayed during demonstrations to better understand the progression of events exhibited on the Ground Truth Display.

2.4 BEM/ANALYST Work Stations

During the running of a full-up BEM simulation (which involves the ANALYST), user interaction is accomplished by means of three work stations:

- BEM Controller Station
- Blue Sensor Control Station (BSCS)
- ANALYST Station

Figure 2-4 shows the current BEM/ANALYST Work Station Configuration. Each of these stations contains a keyboard and graphics display. (See Appendix V for a description of the laboratory configuration.)

The BEM Controller Station controls the simulation itself. The simulation is initiated from the text terminal of this station and can at any time be suspended. During a suspension, the user can change certain kinds of data and even alter sections of code. The simulation can then be restarted and it will take up where it left off.

The Blue Sensor Control Station serves to monitor Blue sensor movement and sensor reports coming from the BEM. This station also serves as a vehicle for dynamically tasking sensors. In the real world, intelligence missions are conducted primarily in an effort to discover information relating to enemy position capability and intention. In the analytical environment, it is inappropriate for the collection manager to be able to view the enemy order of battle. For this reason, the Blue Sensor Control Station is remoted from the BEM. Although its functions belong in fact to several intelligence organizations in the field, it has been useful to combine them at one station for purposes of analysis.

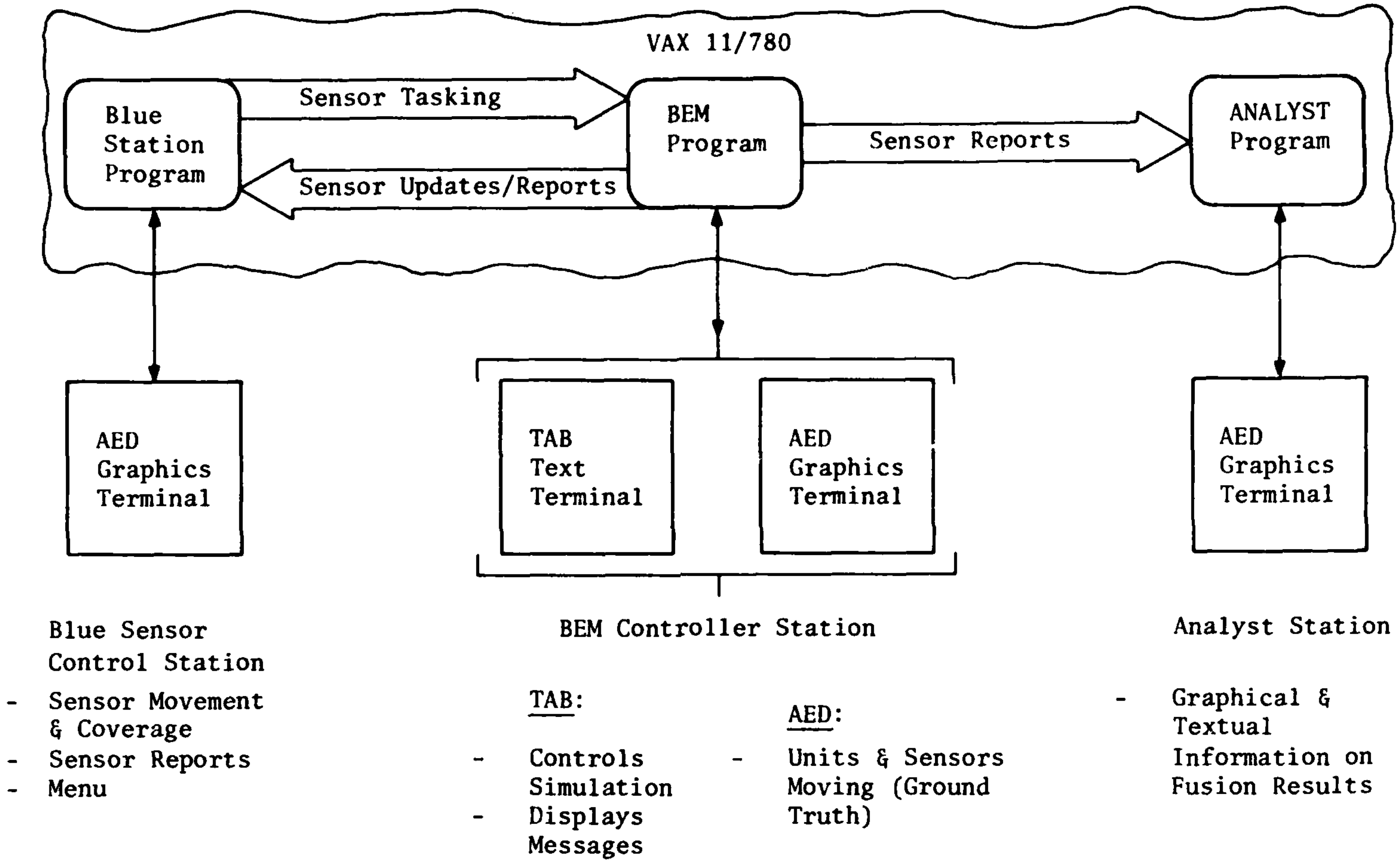


FIGURE 2-4
BEM/ANALYST WORK STATION CONFIGURATION

All of these work stations allow a user to move the cursor to a point on their respective graphics displays and make enquiries as to the nature of an object displayed there.

The next four sections of this report discuss these stations and their displays in more detail.

2.4.1 BEM Controller Station

The BEM simulation is projected graphically onto the Ground Truth Display. This display is comprised of three functional parts: (1) a node or terrain background, (2) the Red threat units, and (3) the Blue sensor units. In addition, the activities of the military units are dynamically displayed. Figure 2-5 shows the terrain symbology used.

2.4.1.1 Background Displays

2.4.1.1.1 Trafficable Path Network Background. Threat and sensor ground units are constrained by terrain features to a limited set of trafficable paths. These paths are represented by 493 nodes in the BEM simulation. Each of these nodes has properties describing its type (city, bridge, obstacle or road intersection), its location, and any neighboring nodes to which there are trafficable paths. The graphics system can portray this nodal network to support development of route selection criteria or to enhance other detailed threat analyses. Figure 2-6 shows the current node network.

2.4.1.1.2 Stylized Terrain Feature Background. Demonstrations of the BEM require a graphical background which has more militarily significant information. Major terrain features such as high speed roads, major rivers, and forested areas need to be displayed, along with large urban centers and key bridges. Here the viewer is able to watch the simulation against an aggregated background where model details are less important than major trends in behavior. This is the default background for the BEM Ground Truth Display. Figure 2-7 shows the terrain display.

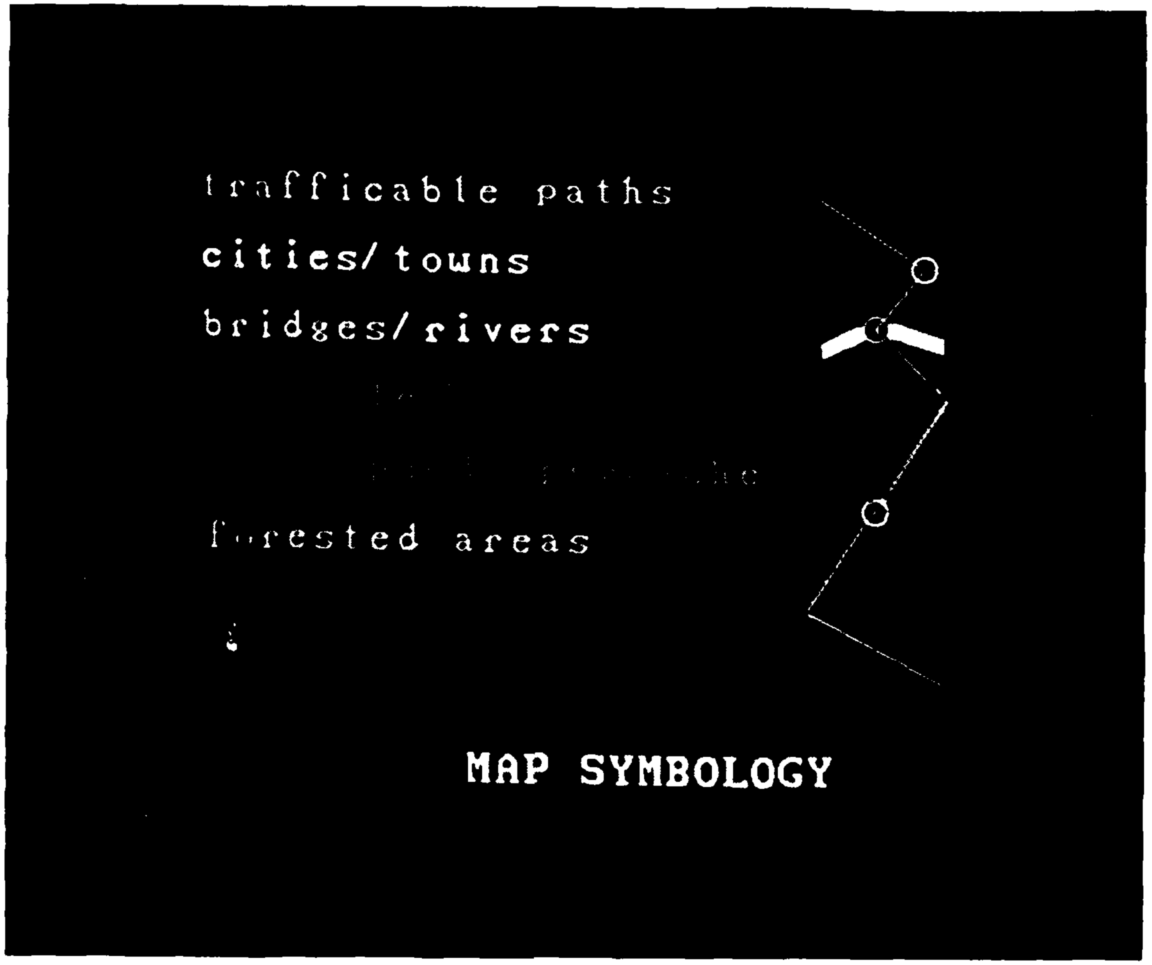

trafficable paths
cities/towns
bridges/rivers
forested areas
MAP SYMBOLOGY

FIGURE 2.5

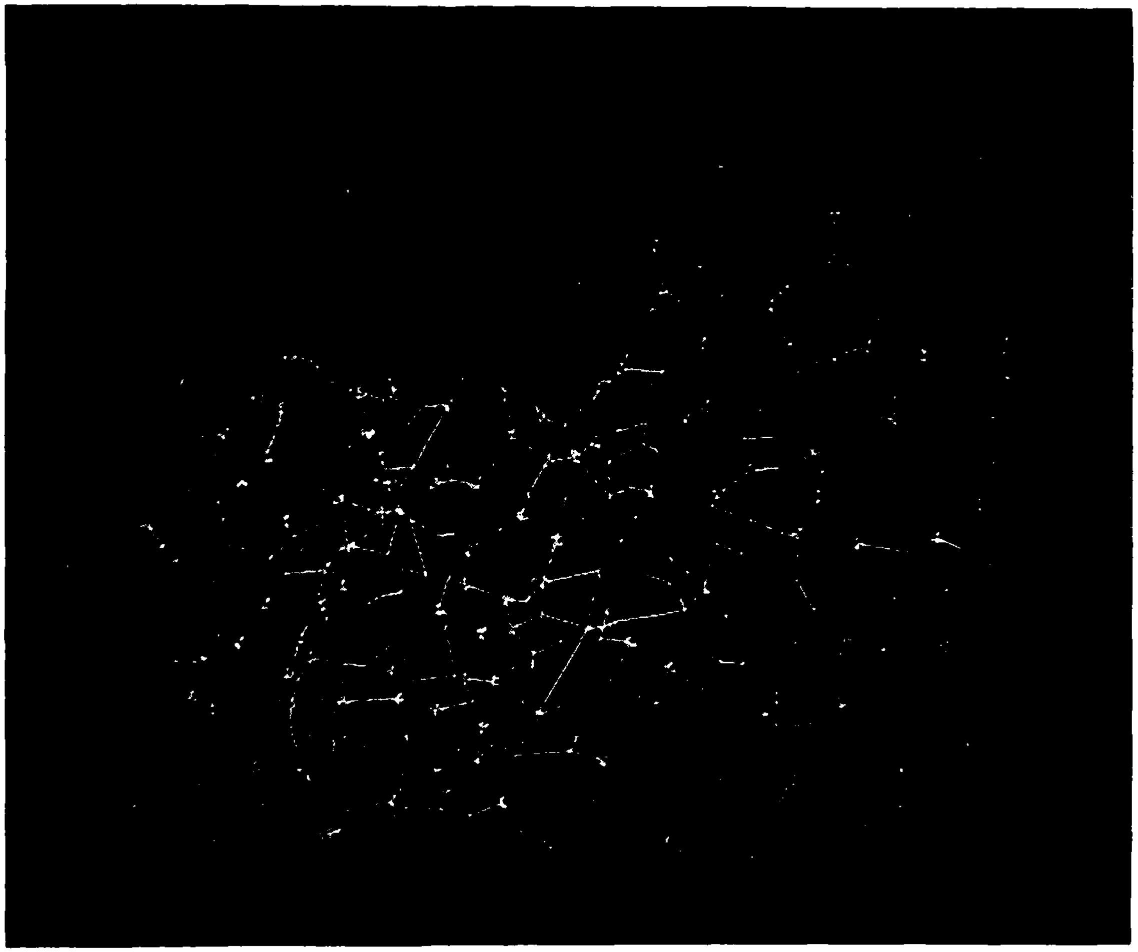


FIGURE 2-6
CURRENT NODE NETWORK

FIGURE 2-7
TERRAIN BACKGROUND

2.4.1.2 Red Threat Units. Threat units in the BEM are of five basic types: tank, motorized rifle, artillery, air defense, and combat service support. These require symbols to reflect both their function and their size, e.g., company/battery, battalion, regiment and division. The symbology used is similar to approved military convention while allowing for rapid recognition in a complex display. Figure 2-8 shows Red threat units representing a division.

2.4.1.3 Blue Sensor Units. As stated earlier, the BEM uses five basic types of sensors: MTI, COMINT, ELINT, IMINT (photo), and CMCB. The symbology used to represent sensors does not distinguish between sensor types but rather whether the sensor is airborne or on the ground. The symbols were chosen to make sensors easily distinguishable from the Red threat units.

2.4.1.4 Enemy and Sensor Activities. The Ground Truth Display dynamically portrays not only the simulation units but also their activities. These are discussed below.

Threat and sensor units that move require a graphics display of that movement. Basically, movement is depicted by erasing a unit's symbol at the current position, drawing a vector to the destination, erasing the vector, and displaying the symbol at the end location.

Military communications (for the threat units only) are represented as radiating concentric circles centering on the transmitting element and colored to correspond with the type of communications currently used in the BEM, i.e., fire call, net call, ready call, control call, sitrep call, and confirm call.

There are four types of threat radars used in the BEM: meteorological, fire control, heightfinder, and surveillance. Each radar "on status" event is represented as the displaying of a radar symbol next to the transmitting unit in a color appropriate to its function.

There are three types of firing events shown on the Ground Truth Display: artillery, missile, and rocket. Each type requires a separate color and is shown as a flashing asterisk near the firing unit.

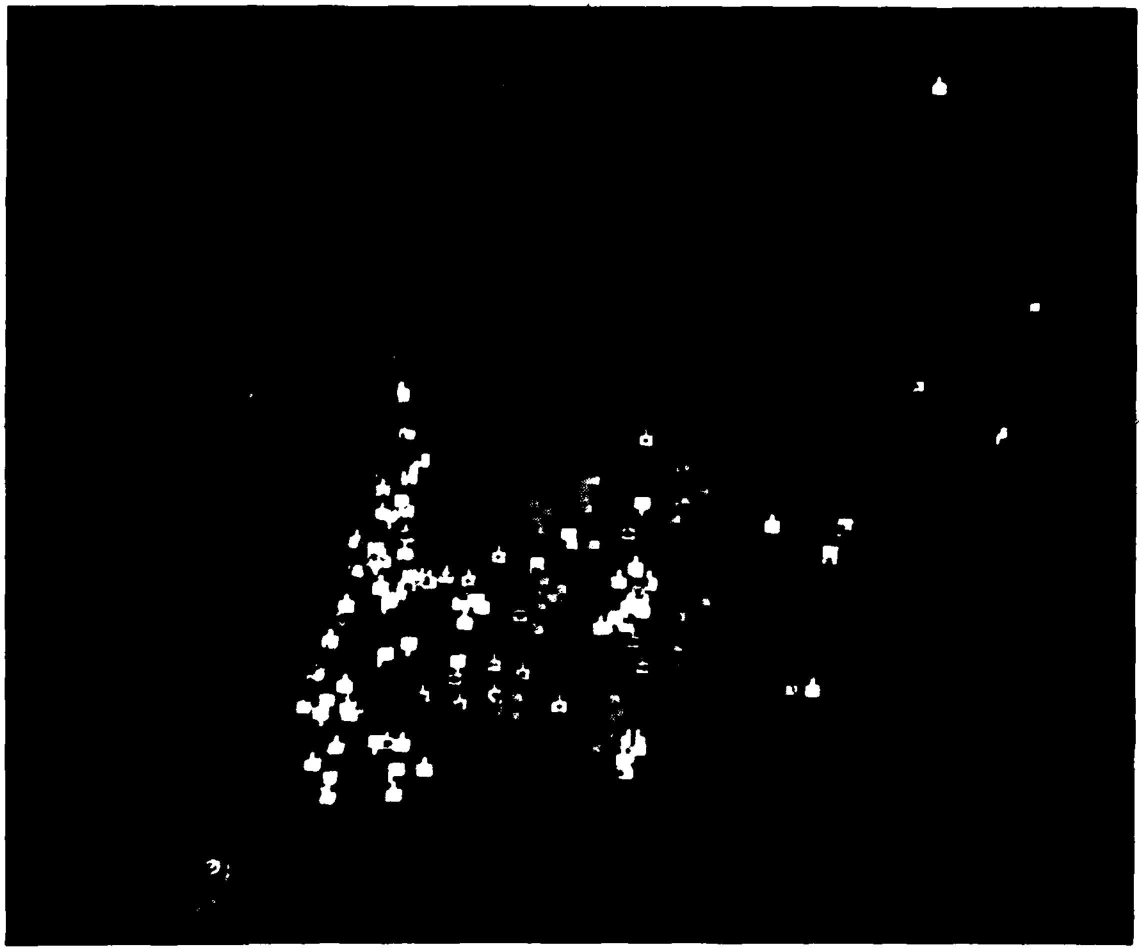


FIGURE 2-8
RED THREAT UNITS REPRESENTING A DIVISION

There are currently three types of sensor coverage displayed on the Ground Truth Display. Airborne photo-imagery is represented by a square with the current sensor location in the center. Omnidirectional sensors (COMINT) require a circular coverage display, while directional sensors (MTI, ELINT, CMCB) require a fan-type coverage display with the origin at the current sensor location. Displays of sensor coverage are used to depict graphically the tactical duty cycle of particular sensors. Figure 2-9 depicts ground and airborne sensors and their coverages over the battlefield.

2.4.2 Blue Sensor Control Station

The Blue Sensor Control Station (BSCS) is a separate process from the BEM, designed to provide an interactive window into the sensor tasking and execution functions contained in the battlefield simulation. Additionally, the Blue Sensor Control Station displays the results of BEM sensor operations in the form of tactical intelligence reports displayed on the graphics screen.

2.4.2.1 Utility. The Blue Sensor Control Station enables the user acting as a collection manager to formulate sensor taskings for each of five types of intelligence sensors. These taskings are then passed to the BEM for implementation. Providing there are available sensors, the BEM orchestrates the performance of the sensor platforms and collection devices. Sensor platform positions and coverage areas are displayed at the Blue Sensor Control Station Display, allowing the collection manager to monitor the execution of earlier taskings. The BEM sends tactical sensor reports to the Blue Sensor Control Station which displays them on the graphics screen. The Blue Sensor Control Station also records reports in history files for use in later evaluations. At any point, the user can interact with the Blue Sensor Control Station display to obtain information relating to a particular sensor or report on the screen. The mission tasking, monitoring, and assessment cycle continues throughout the simulation, and the collection manager at any point can control sensor employment in the BEM through this station.

FIGURE 2-9
GROUND AND AIRBORNE SENSORS AND THEIR COVERAGES
OVER THE BATTLEFIELD

2.4.2.2 Operation Once the BEM is running, the user then loads the Blue Sensor Control Station environment into a ROSS program in the same working area as the BEM and prepares the station with the following command:

(ask station-manager prepare the station)

This behavior is principally graphical in nature. It displays the stylized terrain background and the station menu on graphics screen. Figure 2-10 is a view of the station after this command is completed.

The user can now cause the station to begin looking for sensor position updates and sensor reports put out by running BEM. The command to issue is:

(ask station-manager turn the station on)

At this point the station is running and the user controls its operations with the menu shown in Figure 2-10. By placing the joystick cursor over the desired circle option and pressing a key, the user can select among the choices shown.

The menu shown allows for selective decluttering of the display by alternately showing or hiding parts of the display such as the map, sensor positions or sensor reports, or the menu itself. A joystick is used to "pick" sensors and sensor reports whose detailed information is to be displayed.

Figure 2-11 shows the station during a simulation with several sensors operational, their coverage areas and sensor reports of several types displayed on the screen. The station's interactive querying capability is shown in Figure 2-12, where the user has picked an ELINT sensor report. The user can read the report's property list while the station continues to process incoming sensor position updates and/or additional sensor reports.

2.4.3 ANALYST Station

2.4.3.1 Utility. The ANALYST Station allows the user to interact with the fusion model and depicts on a graphics screen the results of processing sensor reports from the BEM. The ANALYST is an expert system which transforms sensor reports to meaningful hypotheses as to the current state of the battlefield by applying various knowledge-bases.

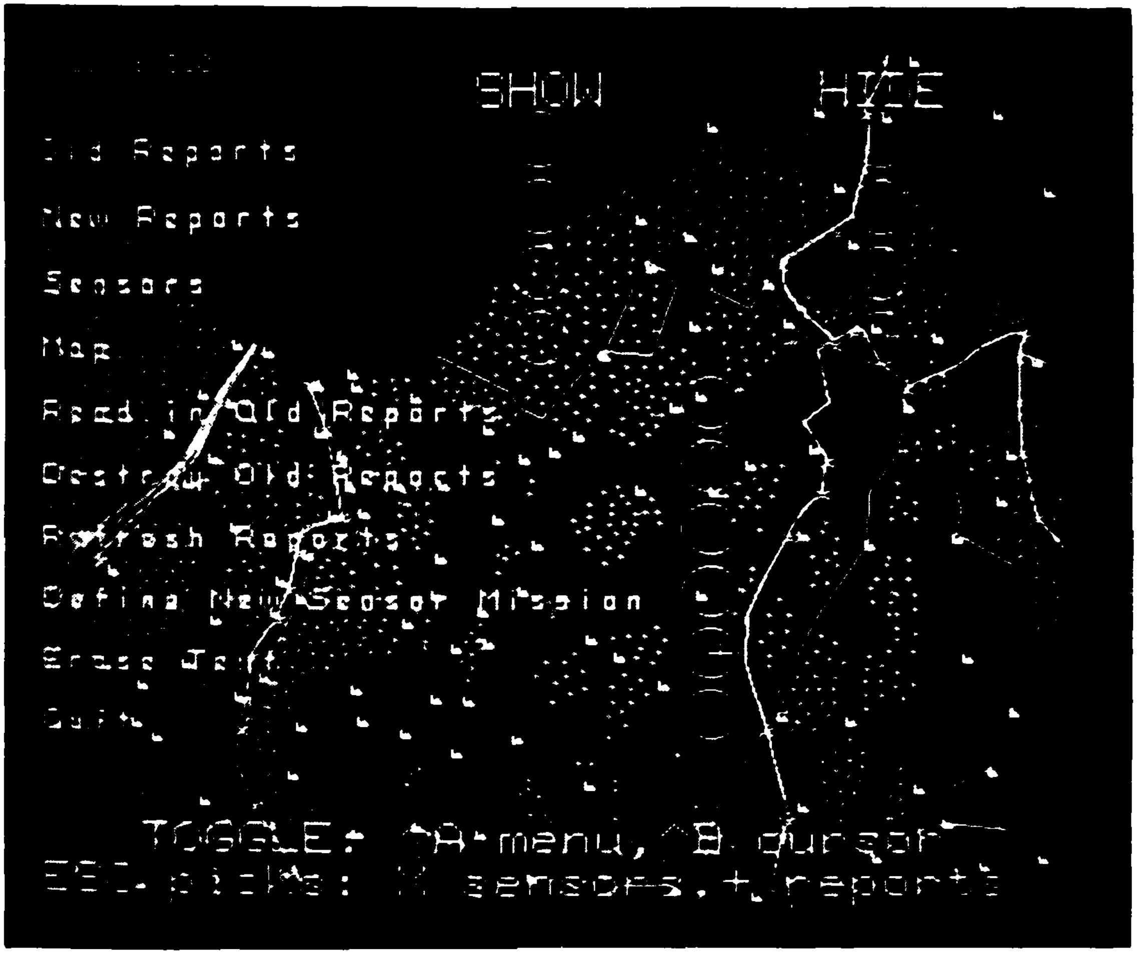


FIGURE 2-10
BSCS – GRAY MAP BACKGROUND WITH MENU OVERLAY

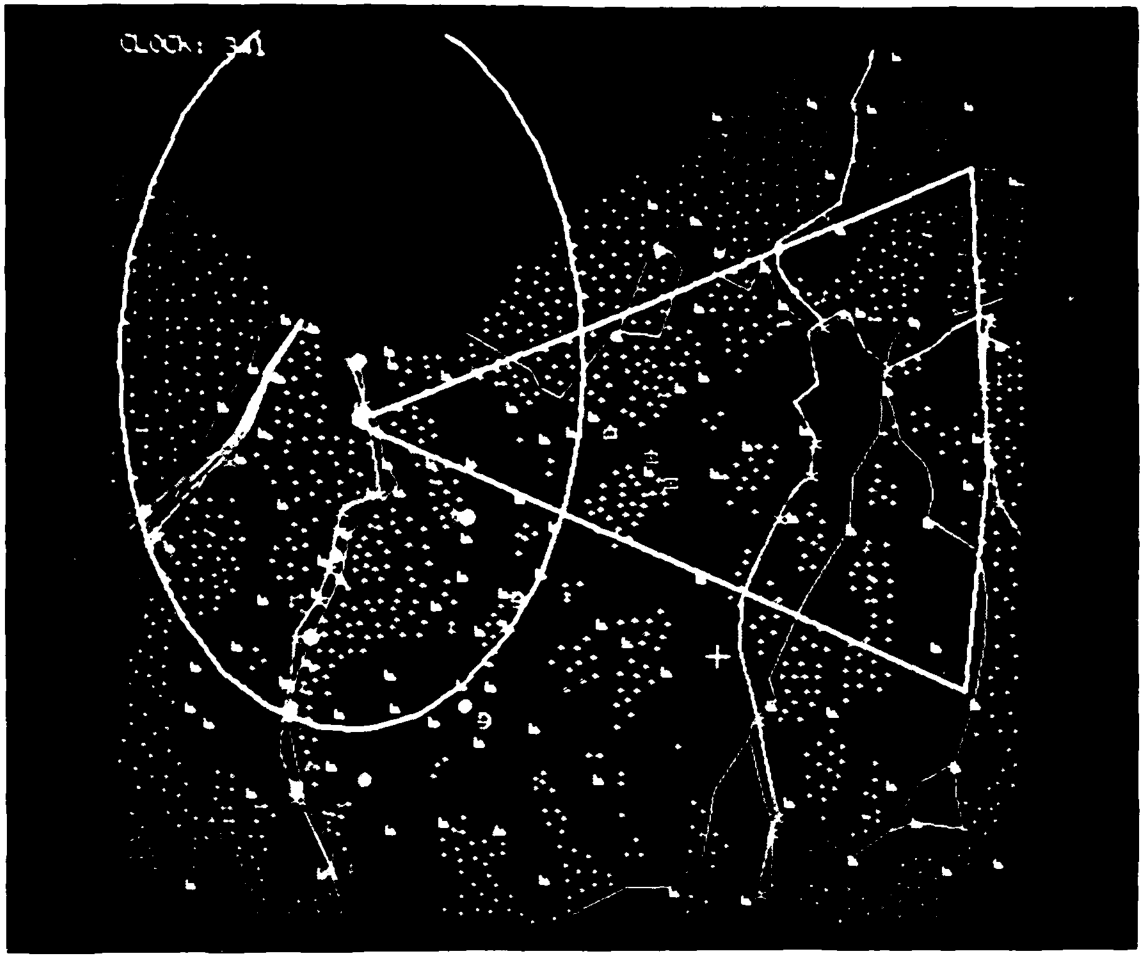


FIGURE 2-11
BSCS – MAP, NO MENU, SENSORS AND REPORTS

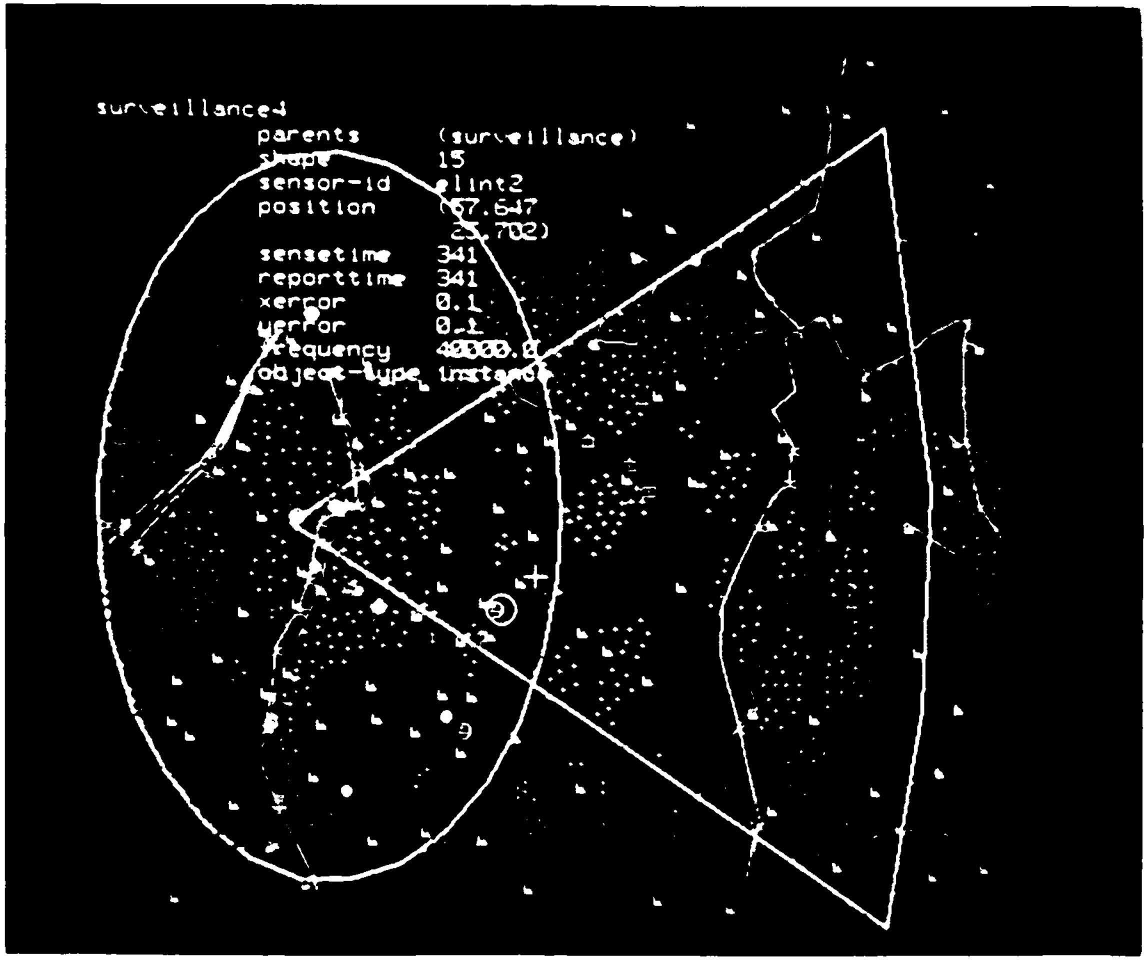


FIGURE 2-12
BSCS – INTERACTIVE QUERYING CAPABILITY

2.4.3.2 Operation. Sensor reports, collected from the BEM, are partioned into five groups corresponding to five types of sensors. Knowledge is applied to each group of sensor reports to produce localized clusters of activity as shown in Figure 2-13. The numbers represent the number of reports making up the cluster, and the color represents the type of activity, i.e., green for radars, red for firings, etc. Pattern knowledge is then applied to these clusters to hypothesize the existence of military entities. If the ANALYST is not sure of an entity at this point, it is shown in pink. Figure 2-14 shows another look at the screen with some military objects determined and others in an uncertain state (i.e., those shown in pink).

General military knowledge is next applied in an all-source process to refine the situation. Some information about known objects may be transferred to those close by which are unknown; initial disparities will be resolved; and enemy doctrine will be examined to approve or disapprove of conclusions already made.

Finally, information both received and hypothesized which is dated will be purged from the system after a certain length of time. Purged units are shown in dark blue.

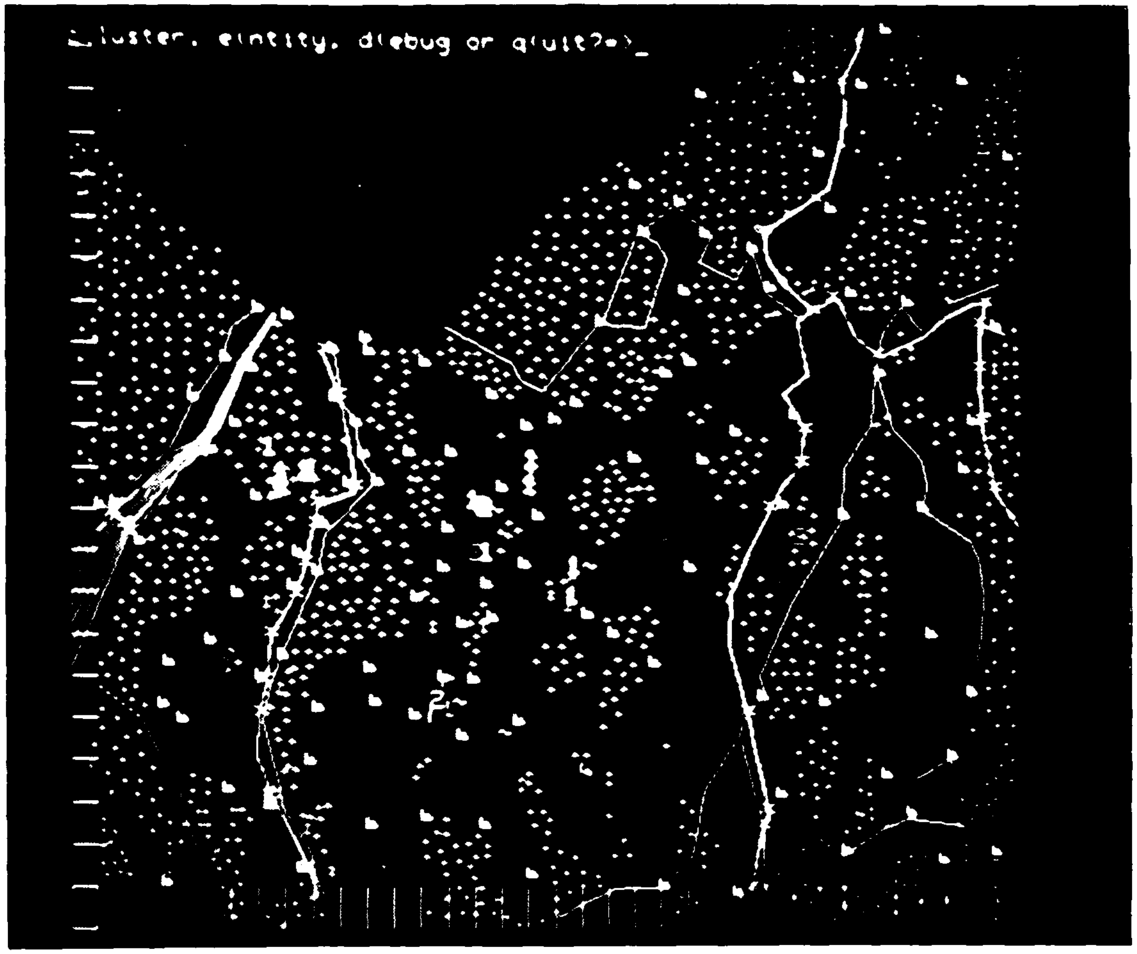


FIGURE 2-13
ANALYST STATION DISPLAY – INITIAL CLUSTER REPORTS

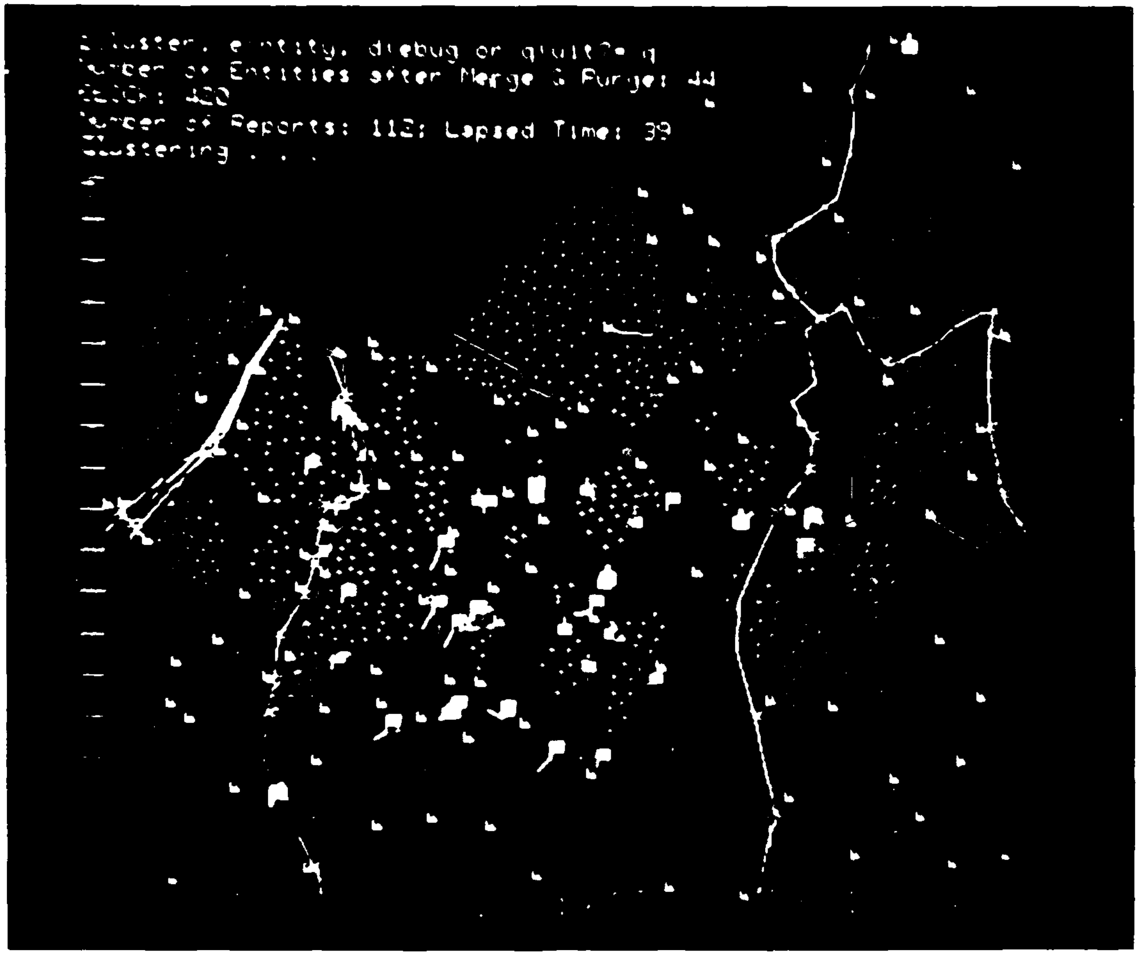


FIGURE 2-14
ANALYST STATION DISPLAY – SHOWING DETERMINED AND UNDETERMINED MILITARY OBJECTS

3.0 DETAILED DESCRIPTION OF THE BEM

3.1 Model Overview

As discussed in Section 1.0, of major importance in the design of the is the use of object-oriented programming techniques. Constructing the with the ROSS language was considered an important test case in determ the applicability of the object-oriented programming to other forms of modeling. For a report on the evaluation of object-oriented programmin Army modeling, see the MITRE Working Paper.[4]

As noted earlier, objects in the BEM can generally be divided into that have some real-world counterpart and those that do not. This l group is referred to as auxiliary objects, which perform computations service to the real-world objects of threat units and Blue sensors. Figur shows the ROSS object hierarchical structure of the BEM. The right column headed by Scenario comprises the auxiliary objects in the BEM. properties and behaviors resident at one level of this hierarchy ca inherited by any actor at a lower level. Conversely, any property or beh at a lower level can overide the same property or behavior resident higher level. Thus, in order to save space, properties and behaviors ir BEM are placed at the highest level of commonality.

The primary purpose of the BEM is to generate realistic sensor rep So that this function may be better understood in the BEM/ROSS environn Table 3-1 portrays the major elements of controlling sensors and of se detection of Red units. A view of the process is given here by stey through the sequence of message passing from the first tasking of a sensoı the ultimate detection of a threat unit. The example uses an airborne E sensor which is capable of detecting emissions from Red unit anti-air radars if those emissions fall within its coverage area. The messages depi are the actual ROSS messages sent.

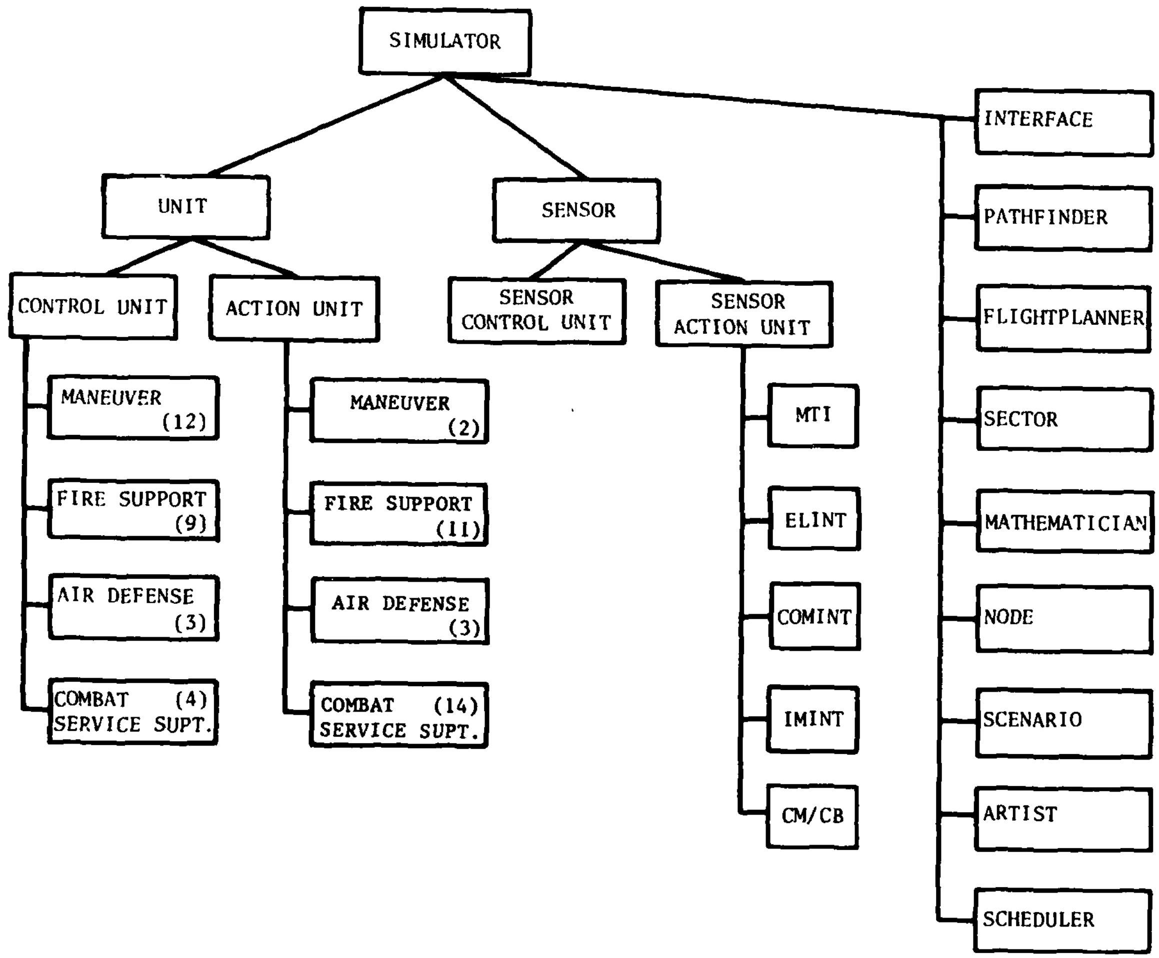


FIGURE 3-1
ROSS OBJECT HIERARCHY IN THE BEM

The actor "Scenario" is the BEM's source of initiating events. The event of interest in the case shown in Table 3-1 is the tasking of an ELINT sensor. This tasking alternatively could have been generated by the Blue Sensor Control Station through interaction with the user.

In the initial message in Step 1, controll, elint, 8, 3, and (50 25) represent the specific values of variables within the message. All other words form the static format of the message. This message is sent to a sensor-control-unit called Controll, which will determine which sensor is available and construct a task and plan for it using the incoming information.

In Step 2, when Controll sends the message, the expressions !task and !plan will be evaluated and sent to the sensor-action-unit indicated by the variable "sensor".

Step 3 indicates that the sensor-action-unit will ask an auxiliary unit called Flightplanner to construct a route for the airborne sensor, and also ask another auxiliary unit, Mathematician, to determine when the coverage area of the sensor penetrates (i.e., turns on in) the global sector (an intermediary between Blue sensors and Red units).

In Step 4, Mathematician will, in response to the previous message, send a message to the global sector telling it what sensor has penetrated it and sending pertinent additional information such as the next time the sensor will turn off.

In Step 5, the sector will ask the Mathematician to determine the time and space overlaps between the ELINT sensor and any Red radars. The result will be a list of the Red radars which are on within the ELINT's coverage area at the same time the ELINT sensor is turned on. Also returned will be the times when these detections first occur. When these times are reached in the simulation, the Sector will send a message to the appropriate sensor to check for sensor detection as indicated.

In Step 6, when this message is received by a generic actor Sensor, the proposed detection is passed on directly to the Interface telling it which

Table 3-1
Major Elements of Sensor Control and Detection

Step	Actor	Example of Message Sent by Actor
1.	Scenario file or Sensor Control Station	"tell controll send up airborne elint sensor in 8 km racetrack pattern 3 times around with initial pt (50 25)"
2.	Sensor-Control-Unit	"tell !sensor move airborne sensor in racetrack pattern with !task and !plan"
3.	Sensor-Action-Unit	"ask flightplanner move airborne sensor !myself using racetrack pattern"
		"ask mathematician determine penetration of global generic sector with airborne sensor !myself"
4.	Mathematician	"tell sector !sensor has penetrated the sector with !sensor-data"
5.	Sector	"ask mathematician determine time/space overlaps between elint !sensor-data and any red radars"
		"tell !sensor check for sensor detection of !unit of type !type and frequency !frequency"
6.	Generic Sensor Type	"tell interface !myself has detected !unit at !frequency and !duration"
7.	Interface	Extracts from the database the type of information that particular sensor would report and sends it (if required) to Blue Sensor Control Station or ANALYST

sensor has detected what threat unit and relaying the detected emission frequency and duration.

In Step 7, the Interface will extract from the unit's database only the information that a sensor could realistically detect. These sensor reports can then be sent to ANALYST and/or the Blue Sensor Control Station.

3.2 Actors - Actor Files in the BEM Simulation

The following sections of the report discuss the current actors in the BEM simulation by describing their purpose and their major behaviors.

3.2.1 Simulator

The Simulator is the highest-level actor in the object hierarchy. Created by the ROSS actor Something, all other actors are directly or indirectly created from Simulator. A well-defined inheritance tree is thus established even though inheritance features are not used for all actors. Simulator is the principal BEM object with which the user can control the simulation. Prior to the beginning of a simulation, the user sends a message to the Simulator by typing:

(ask simulator prepare simulation using scenario xyz).

Upon receipt of this message the Simulator will:

- override certain of its defaults with those in the scenario file named xyz
- ask the graphics generator (the Artist) to draw the appropriate terrain background on the Ground Truth Display
- ask the Artist to draw all of the military units on the Ground Truth Display in their initial positions.

When this is completed, the user types:

(ask simulator start simulation).

When the Simulator receives the "start simulation" message, it:

- tells the scenario actor xyz to "prepare to start simulation" which initiates all preprogrammed events

- o sets up a loop which will loop once for every simulation time segment (tick) required to reach the designated end of the simulation.

Within the loop for every simulation tick, the Simulator will:

- o check to see if an interrupt has occurred which is the user's method for halting the simulation
- o ask the Artist to print the simulation time on the Ground Truth Display
- o ask the simulation clock to advance one tick
- o cause all sensor reports accumulated during the tick to be sent out to ANALYST and/or the Blue Sensor Control Station.

The above two commands are the ones required from the Controller Station to run the BEM simulation.

The user can also send the Simulator messages to start or stop:

- o all graphics on the Ground Truth Display
- o sending sensor reports to ANALYST and/or the Blue Sensor Control Station.

Finally, by sending the Simulator the message "prepare simulation again," the user can cause a complete restart of the simulation from the beginning without reloading the system.

3.2.2 Scenario

The Scenario actor is created by the Simulator and takes a name related to a simulation being shown. This actor contains property information and messages which will govern how the BEM simulation will be run. The graphics and timing defaults listed as properties of the Simulator can be overridden by the corresponding properties of the Scenario. The Scenario has only one behavior. In response to the message "prepare to start simulation," the scenario will send messages which initiate movement, artillery and air defense firings, radar emissions and sensor taskings. These messages will be sent at specified times during the simulation.

3.2.3 Artist

The Artist is an auxiliary object which produces the graphics on the Ground Truth Display. It provides a high-level ROSS interface so that the BEM actors need not concern themselves with the details of graphics production. In the BEM, medium-level BEM related graphics routines reside in an ARTFNS.L library while BEM- independent, low-level graphics instructions reside in the AED.L file.

3.2.3.1 The Low-Level Graphics Device Driver (AED.L). The device currently used in the SPL for graphics displays is the AED 512 graphics terminal. As in the case of any graphics terminal, it has its own set of firmware which recognizes specific display commands to produce images on the display monitor. In an effort to make the Artist as device independent as possible, these functions were written directly in LISP and placed in a file separate from the ROSS files for ease of compilation and speed. Other SPL LISP-based systems also use this file as the device driver for the AED, as in the case of the VAX-based ANALYST. This file is application independent and represents a LISP graphics library for the AED terminals. An example of a LISP function used is:

(draw-circle 10)

which draws a circle with the origin at the current cursor position in the current color with a radius of 10 pixels. This function generates a character stream which drives the AED firmware through an RS232 interface to produce the circle.

3.2.3.2 Artist Messages/Behaviors. When receiving the ROSS message

"display node network"

the Artist accesses the BEM node network data structure to produce an image on the background plane shown in Figure 2-6. This display uses the colors gray, blue, and black. The cities are represented as blue circles, the bridges or obstacles as gray circles, and the roads as gray arcs. Black is reserved to enable erasure which is available by substituting the word "erase" for "display" in the invoking message pattern. The substitution of "erase" for "display" is allowed in all Artist messages.

When receiving the ROSS message

"display tic (or grid) marks"

the Artist will cause kilometer-spaced tic or grid marks to be drawn as an aid in the determination of points of reference. Tic marks are spaced along both the x and y axes. Grid marks cross the entire screen along both axis. Both tic and grid marks are drawn in gray.

When receiving the ROSS messages

"display towns and bridges" and "display terrain features"

the Artist draws the stylized terrain display as shown in Figure 2-7. Forested areas, major highways, cities and bridges are drawn with these messages where tree symbols are green, roads and cities light blue, and bridges gray.

When receiving the ROSS message

"place > shape at vicinity"

the Artist will draw an object symbol corresponding to the value of "shape" at the location specified by "vicinity". All threat units and sensors are initially displayed with this command.

When receiving the ROSS message

"move > actor from > location to > destination using > shape"

the Artist erases the specified shape at the specified location, draws a line in the appropriate color to the specified destination, erases the line, and places the symbol at the destination to give an appearance of object movement. The actor variable holds the actor name involved, and although this information is not currently exploited, it aids considerably in tracing the Artist behaviors.

When receiving the ROSS messages

"display >type comm at >location"

or "display >type radar at >location"

or "display >type burst at >location"

the Artist will draw special symbols at the specified location representing one of these activities. For the "comm" request, the Artist will display and then automatically erase four concentric circles in an appropriate color to indicate a tactical communications event. For the "radar" request, the Artist will display an appropriately colored radar symbol to the left of the location specified. For the "burst" request, the Artist will display an appropriately colored asterisk to the left of the specified location. Because this depicts a firing event, the symbol will be erased automatically.

When receiving the ROSS message

"display > actor sensor coverage"

the Artist will display the particular sensor's coverage area using the sensor's range, current location, and coverage shape (circle, fan, or square). All of these shapes are orange colored and will be appropriately clipped at the edges of the Ground Truth Display.

When receiving the ROSS message

"update the clock to > time"

the current value of simulation time is displayed at the upper left hand corner of the screen.

The ROSS message

"correspond"

enables user interaction with the Ground Truth Display screen for information query. The Artist initiates a dialogue with the user on the text terminal of the Controller Station concerning the objects on the Ground Truth Display. The user may request information about nodes, sectors, threat units, and sensors, by "picking" them on the graphics terminal with the cursor. The Artist will then list on the text terminal the full property list for that item.

3.2.4 Node/Node Environment File

The Node Environment file is a file of node instances created by the generic Node object. In this file the individual node properties - position, associated sector, node type and linked neighbor nodes - take on specific values. Nodes do not have behaviors. They exist in the current BEM as a convenient, easily accessible database. The 493 node instances create a node network corresponding to the road network depicted in the previous chapter. The property "linked-nodes" is a list of other nodes to which there exists a trafficable path. This information is used by path construction routines to generate routes for the movement of ground units, both threat and sensor.

3.2.5 Pathfinder

Pathfinder is an auxiliary object created at the instance level. It is used both in the preprocessing of input data and dynamically in the simulation to find paths for the Red units and Blue ground sensors over the trafficability network. The Pathfinder accesses unit and network data and determines a time-ordered route using pathfinding and other algorithms written in LISP.

The Pathfinder does not have any properties but has three behaviors:

- o To determine a time-ordered network path for a Red unit
- o To determine a time-ordered network path for a Blue ground sensor
- o To move either a sensor or unit over the network path at the predetermined times.

3.2.6 Scheduler

Scheduler is an auxiliary object created at the instance level. The purpose of Scheduler is to create those events which are either of a random nature or would be created by the interaction of opposing units in a two-sided simulation. Events currently modeled are as follows:

- o Artillery fire missions
- o Air defense engagements
- o Radar on-off status.

Scheduler has the following properties which control the frequency of events:

- o Preparation-frequency, the mean frequency with which artillery preparation fire missions are generated
- o Fire-mission-frequency, the mean frequency with which artillery target-of-opportunity fire missions are generated
- o Firing units, a list of the artillery fire units available depending on the type of mission

- Artillery preparation start-time, stop-time and length, set by the initiating message
- Air defense engagement frequency.

The Scheduler has eight behaviors described below:

- "Initiate call for fire." This causes the Scheduler to take three actions:
  - schedule the next "initiate call for fire" according to an exponentially distributed random variable time in minutes with mean of the fire mission frequency
  - randomly pick one of the maneuver companies in contact to call for artillery support
  - send a message to that company to make the call for artillery.
- "Cease call for fire." This causes the Scheduler to unplan its next "initiate call for fire" and thereby terminates the cycling until it is reinitiated.
- "Conduct > t-length minutes fire starting at time >t-start." This causes the Scheduler to set its property values for the preparation start-time, length and stop-time and to call itself to "fire the preparation."
- "Fire the preparation." Upon receipt of the message, the Scheduler takes the following three actions:
  - reschedules the next "fire the preparation" message to itself unless the preparation stop-time has been passed
  - randomly picks an artillery battery which is available to fire (not moving or otherwise engaged)
  - sends a message to the unit to fire.
- "Cease fire the preparation." This causes the Scheduler to unplan "fire the preparation" and thereby stops the cycling of calls until reinitiated.
- "Initiate Red radars." This causes the Scheduler to
  - schedule a "create air defense acquisition" message according to an exponential distribution with a parameter of the air defense engagement frequency

- schedule a duty cycle for each of the various units with radars (See section 3.2.8 for more description of the SA-6 acquisition sequence).

o "Create air defense acquisition." This message causes the Scheduler to

- reschedule another "create air defense acquisition" message to itself

- check each air defense battalion and if one is found that has its surveillance radar on, the Scheduler sends a message to that unit telling it it has acquired a target.

o "Cease air defense acquisition." This message causes the Scheduler to unplan "create air defense acquisition" and stops the cycle until it is reinstated.

3.2.7 Sector/Sector Environment File

The Sector is an auxiliary object which performs two functions for a BEM simulation:

o It acts as a third party between sensors and threat units.

o It can localize the search for threat units by some sensors.

The battlefield is divided into rectangular sectors which do not correspond to any military sectors. These sectors are all created as ROSS instances and are defined in the Sector Environment file. There are no behaviors in this file; the only data slot is for the position of the sector instances. The position is given in the form of four x-y locations of the corners of the sector. Figure 3-2 shows the allocation of sectors over the battlefield in the current BEM.

Sector behaviors reside in the generic Sector actor, referred to as the global sector when the entire battlefield is treated as one sector. When this occurs, the global (generic) sector acts as an instance would, and messages are sent directly to it instead of the actual sector instances.

3.2.7.1 Sector as Third Party. All five sensor types use the sector as a third party between them and the threat units. This is done to increase the integrity of the simulation since it is not appropriate to have sensors accessing threat unit information. Since the sector can know the dispositions

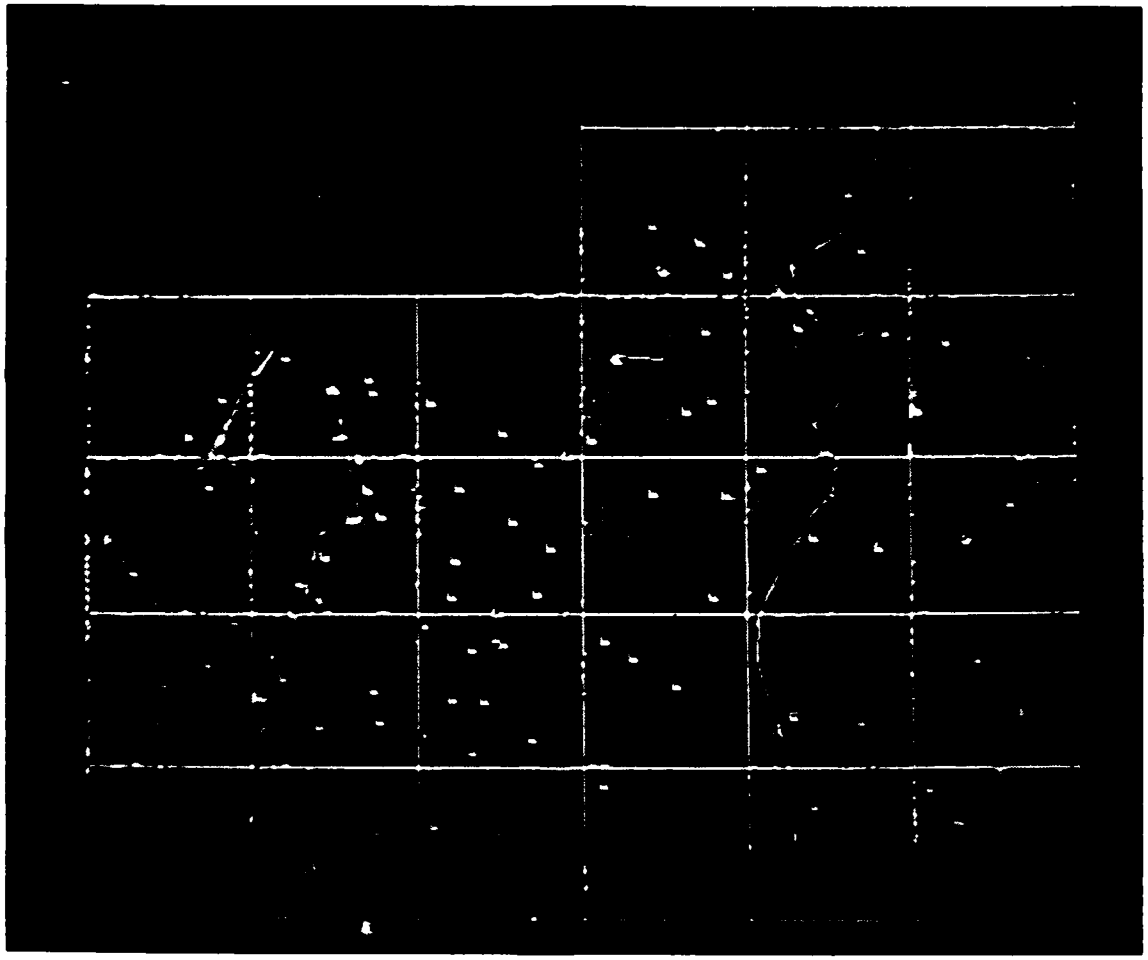


FIGURE 3-2
SECTORS IN THE BEM

of both sensors and units, it acts to complete the sensor detection process once an initiating event has occurred.

Table 3-2 shows the events which trigger the detection process in Sector. The occurrence of any of these events results in a corresponding message being sent to either the global sector or a sector instance. Thus there are seven behaviors within Sector.

The time duration of events is crucial to determining how the sensor detection process works. Even though voice communications, artillery firings, and picture-taking do have time lengths associated with them, these times are considered small enough so that the event can essentially be considered to occur at a single point in time. Therefore, when voice communications occur, the global sector will receive the ROSS message

" unit has voice communication with
frequency > freq and duration > duration"

and when artillery or missile firings occur, the global sector will receive the message

" unit has fired > artillery-missile".

The response to these messages is essentially the same. The Mathematician is asked to search the list of relevant sensors in order to determine time/space overlaps between sensors and units. For each sensor having a time/space overlap with the unit, a message will be sent to it telling it to "check for sensor detection..." This is the essence of what the Sector does, although the time/space overlap computations in the Mathematician vary widely.

Sector responds to IMINT events upon receipt of the message

" sensor has taken picture at > point"

The response is similar to that described above except that in this case the sensor originated the event, and Sector will check all units within the relevant sensor coverage area.

Because enemy anti-aircraft radars have long duty cycles compared to radios and because movement of threat units through sectors is continuous, detection of these events is dependent on the time intersection between

Table 3-2

Sensor/Threat Unit Events Which Trigger Detection Process

Event	Triggered By	Time Duration	Applicable Sensor
1. Voice Communication	Threat Units	nil	COMINT
2. Artillery Firings and Missile Firings	Threat Units	nil	CMCB
3. Airborne IMINT Photo	IMINT Sensor	nil	IMINT
4. Turn on Anti-Aircraft Radar	Threat Units	Significant	ELINT
5. Turn on ELINT Sensor within Sector	ELINT Sensor	Significant	ELINT
6. Movement into/through a Sector	Threat Units	Significant	MTI
7. Turn on MTI Sensor within Sector	MTI Sensor	Significant	MTI

sensor and unit events. Therefore, for these kinds of detections, the Sector must respond to both the threat unit event and the sensor turning on. For ELINT detections the threat and sensor related messages to Sector are respectively

" unit has turned > type radar on at frequency > freq
and duration >duration"

and " sensor has penetrated the sector with sensor-data".

For MTI detections the threat and sensor related messages to Sector are respectively

" unit has penetrated sector with >unit-data

and " sensor has penetrated the sector with >sensor-data".

Sector has four actor property lists which relate to the four messages above:

- o Active-Red-radars
- o ELINT-sensors
- o Units
- o Sensors (for MTI).

Every time Sector receives one of the four messages above, it adds the threat unit or sensor to its relevant property list and plans to remove it from the property list when that unit or sensor leaves the Sector (or stops moving, etc.). Thus at any given time, all ELINT and MTI sensors active in the Sector and all units which they could detect are known to the Sector. A time overlap is simply that time intersection when both sensor and unit are active in the Sector. The Mathematician is asked to determine this and for those sensors/units which have time overlaps, a second check for space overlaps will be performed. If time/space overlaps exist, the relevant sensor will be informed.

3.2.7.2 Sector for Localized Computation. The second use of sectors is for localization of computations. Because the number of sensors and events for COMINT, CMCB, IMINT and ELINT are relatively few, localization (or use of the sector instances instead of the global sector) is not employed for these sensors. Only MTI sensors can use localization since there are a large number of threat units spread out over the battlefield, and most of them are capable of movement. The idea behind localization is that it is not necessary to check units over the entire battlefield if the sensor covers only a small part of it. Since the sensor coverage for airborne MTIs is extensive, there is no advantage to using the smaller sectors. Consequently, only ground-based MTIs use the smaller sectors.

3.2.8 Flightplanner

Flightplanner is an auxiliary object which constructs routes for the airborne sensors to fly. It contains two behaviors—one which constructs a racetrack pattern and the other which constructs a route based on a list of arbitrarily specified points (used only for IMINT sensors). The relevant ROSS messages to which the Flightplanner responds are, respectively:

"move airborne sensor >sensor using racetrack pattern"

and "move airborne imint sensor >sensor along points> points".

For the racetrack pattern, information relating to the leg length, number of times around, and initial point have been placed in the sensor's "plan" property. This plan will be reworked to form a list of points and associated times which establish a template for the racetrack pattern. The first and last points on the route are the airport. The initial timing of the route is set so that an aircraft takes off as soon as possible, i.e., after an appropriate response delay. The initial point is the lower right hand point of the racetrack pattern. A total of six points will be calculated based on the initial point, and associated times will be calculated using the speed of the aircraft. The aircraft always flies around the racetrack in a counterclockwise direction. Figure 3-3 shows the route constructed for an example sensor with a fan-shaped coverage area. For the template points on revolutions beyond the first, times will simply be updated with the time it takes to make a complete circuit of the racetrack pattern. When the template plan is finished, Mathematician is asked to "calculate update points for !sensor". This process will generate intermediate points according to the resolution specified by the user for use with the graphics displays of the Ground Truth and Sensor

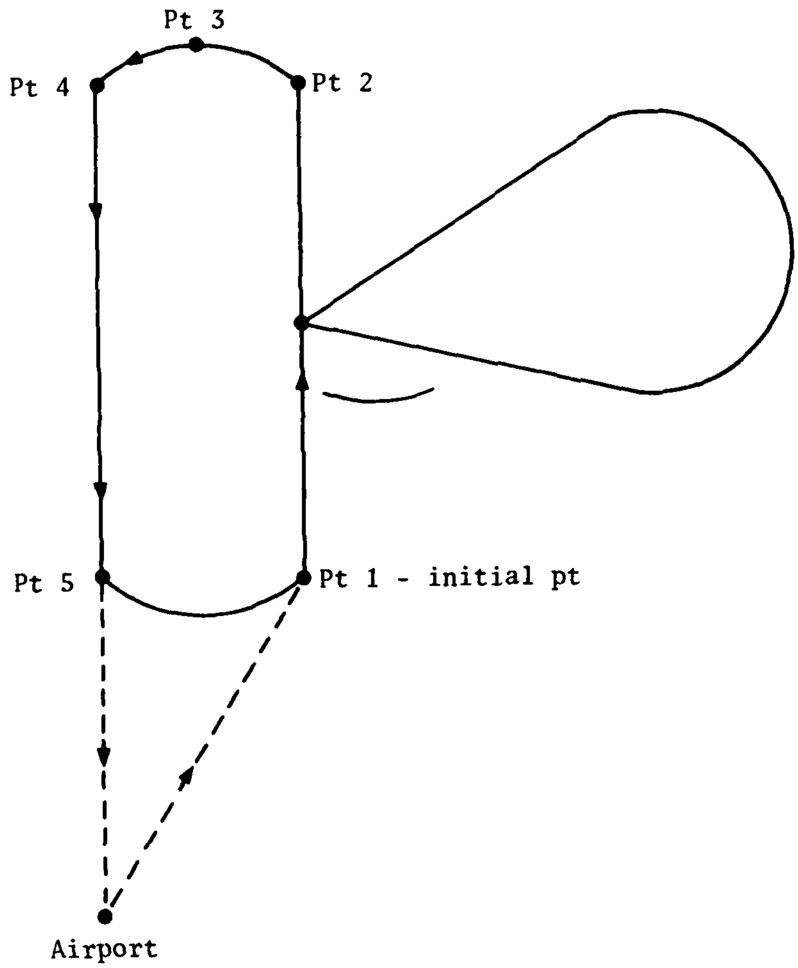


FIGURE 3-3
EXAMPLE RACETRACK PATTERN
CONSTRUCTED BY FLIGHTPLANNERS

Control stations. This updated plan is stored back into the "plan" property of the sensor.

The routes for IMINT sensors are constructed from the list of points passed in the message. Times at these points are calculated using the aircraft's speed and, as before, Mathematician is asked to refine the route by calculating intermediate times and points.

3.2.9 Mathematician

The Mathematician is an auxiliary object which serves the computational needs of the other objects.

The twelve behaviors of the Mathematician are divided into three categories:

- o Determination of time/space overlaps
- o Calculation of sector penetrations by units and sensors
- o Refining stored data.

The determination of time and space overlaps was first described in the section on the actor Sector. The messages are all of the general form "determine time/space overlap between > a and > b," where "a" is a sensor, and "b" denotes the list of relevant units to be searched. If the event is a type that has no time duration (as discussed in Section 3.2.7), a simple determination of space overlap will be made which entails calculating whether a unit is within a sensor's coverage area at the given time. If the event has significant time duration, a calculation will first be made to determine the time overlap. A time overlap is simply the intersection of the time intervals of the unit and the sensor within the sector. Using lower-level LISP routines, further calculations are made to determine if there was a space overlap within that time overlap.

The second major function of the Mathematician is the calculation of sector penetrations by sensors and units. For sensors, penetration applies not to the sensor itself but to its coverage area. A sensor coverage area "penetrates" a sector either when the sensor first turns on or during movement when the coverage area first intersects a sector boundary, and when the sensor coverage area exits the sector. In the case of units, sector penetration means movement of the unit both into and out of the sector. If the unit was already in a sector and not moving, the initiation of its movement would constitute a sector penetration.

Finally, the Mathematician can refine existing data. One method, discussed in Section 3.2.8, was the calculation of intermediate time and position points for a more roughly laid out plan. These calculations are done solely to smooth out sensor and unit movement on the graphics displays. A second form of data refinement occurs when the Mathematician receives the message:

"reconstruct cycle definition for > object".

It then takes information on a sensor's mission start-time, mission duration, and duty cycle and creates a list of sublists which contain on/off times for the entire duration of the mission. The information, put in this form, is used by a variety of other actors.

3.2.10 Interface

Interface is an auxiliary object which acts on messages from the sensor actors (MTI, COMINT, CMCB, ELINT, IMINT) to the effect that a specified unit has been detected. The messages coming from the five sensors are:

"mti >sensor has detected >unit"
"comint > sensor has detected > unit at frequency > freq and duration duration"
"cmcb > sensor has detected > unit"
"elint > sensor has detected >unit of type > type and frequency >freq"
"imint > sensor has detected >unit at time >time".

Some of these messages pass variables relating to frequency or duration since these can only be known at the time a unit initiated the event resulting in its detection. Interface exists to combine these data with other types of data stored in the threat units' data base. The information will be formatted into a sensor report which contains only that information which that sensor could reasonably detect. Sensor reports are accumulated and sent out to the

ANALYST and/or Blue Sensor Control Station by the Simulator at a frequency related to the simulation clock step size.

3.2.11 Unit

Unit is a high-level generic actor with few assigned property values but which has most of the behaviors presently provided to any of the Red units in the simulation. Its parent in the object hierarchy is the generic actor Simulator, its subordinates are Action-Unit and Control-Unit. The other property slots which are assigned values are times such as communication delay time, response delay, fire mission call time and other lengths of communications.

There are five unit behaviors stored at this level of the actor hierarchy. These are behaviors in response to the following messages:

o "Proceed to > position". This causes the unit to take the following actions:

  - notify the Artist to move the unit position on the Ground Truth Display
  - set the unit position to the next position
  - shut down its surveillance radar if it has one
  - set the unit activity to moving.

o "Move according to plan." This causes the unit to take the following action:

  - notify the Artist to display a confirmation communications call
  - notify the Sector of the communications call
  - request the Pathfinder to move the unit from its present position to its destination.

o "Have arrived at node > node." This causes the unit to take the following actions:

  - notify the Artist to display a control call communication
  - notify the Sector that a voice communication has been made and letting it know of the radio frequency and length of call.

o "Have arrived at final destination." This causes the unit to take the following action:

- notify the Artist to display a closing call communication

- notify the Sector of the communication, frequency and directives

- set its activity to "in-position"

- send a message to Scheduler to begin recycling of the units radar if it has a surveillance radar.

o "Turn >type radar >onoff." If the unit is moving, then it sets its radar status to off, unplans any future planned radar cycle, and tells the Scheduler to unplan any cycling. If the unit is not moving and the message is to turn the radar type on, then it takes the following actions:

- notifies the Artist to display the radar symbol

- notifies the Sector that the radar has been turned on and states the frequency and duration

- sets the radar status to on.

If the unit is not moving and the message is to turn the radar off, then the unit makes the notifications to the Artist and Sector and sets its radar status to off.

3.2.12 Control-Unit

Control-Unit is the generic actor from which all the control units (i.e., those with subordinates in the military hierarchy) stem. The actor parent of Control-Unit is the generic actor Unit, while the subordinates are the types of control units, e.g., tank battalion headquarters, tank regiment headquarters, division headquarters.

The properties for the Control-Unit are inherited from its parent, Unit. Control-Unit has four behaviors. The first is for all control units; the second is for maneuver (tank or motorized rifle) battalion headquarters; the third and fourth are for an air defense regimental headquarters.

o "Implement battle plans." This message comes initially from the simulation scenario to the division headquarters and is passed on through the military chain of command until reaching the resolution level, the individual companies in the simulation. On receipt of the message, Control Units take the following actions:

- issue a confirmation call acknowledging receipt of the message
- issue a communication call to each of its subordinates to "implement the battle plan"
- notify both the Artist and the Sector of the above communication
- ask Pathfinder to move the unit according to its pre-arranged movement plan.

The behavior for the maneuver battalion command observation post (COP) is in response to the message "request fire support" from one of the subordinate companies. This simulates the joint effort of the maneuver battalion and the collocated artillery representative. The actions taken are as follows:

- o A communication is made acknowledging receipt of the message, and the Artist and Sector are notified accordingly.
- o A check is made of the artillery battalions available to support that maneuver unit; the artillery must be within range to provide support, and must not be moving or already engaged in firing.
- o The number of volleys of fire needed for the target is generated using a Poisson distribution; this determines both the number of communications to the firing unit for adjusted fire and the number of firings by the artillery unit for that mission.
- o The Artist and Sector are notified of the communications calling this artillery unit to fire.

The first behavior for the air defense regimental headquarters is in response to the message "target acquired" which is generated by the Scheduler to simulate an airborne target being detected by the regiment surveillance radars. The following actions are taken:

- o the regiment headquarters turns its heightfinder radar on and notifies the Artist and Sector accordingly
- o the regiment headquarters picks one of its batteries to engage the target, the battery must not be moving or already engaged in firing
- o a communication is made to the battery to "engage the target" and the Artist and Sector are notified accordingly.

The second behavior for the air defense regiment headquarters is in response to the message "mission complete" when it receives a message from one of its batteries that a mission has been completed. The following actions are taken:

- o The headquarters issues a communication acknowledging receipt of the call and notifies the Artist and Sector accordingly
- o The headquarters sends a message to itself to turn off its heightfinder radar.

3.2.13 Action Unit

Action-Unit is the generic actor from which all the company level units (those without subordinates) stem. The actor parent of Action Unit is the generic actor Unit; the subordinates are the types of action units, that is, the generic company level units.

Five behaviors are stored with Action Unit. The first deals with the battle plan implementation and is used by all action units; the second is for maneuver companies requesting artillery support, the third is for the artillery batteries providing the fire support, the fourth is for air defense batteries engaging a target, and the fifth is for artillery units firing preparatory fires.

The message "implement battle plan" is received from the units military superior and, if the unit is scheduled to move during the time slice, the Pathfinder is asked to move the unit according to the plan, on the units property list.

A maneuver unit in contact may receive a message "call for fire" from the Scheduler telling it to call for fire support as if the unit were engaging an enemy target. The unit sends a communication to its battalion headquarters requesting artillery support; the Artist and Sector are notified of this communication.

A behavior for an artillery battery is in response to the message "fire mission" which it may receive several times, simulating additional calls to adjust effects on the target. The following actions are taken:

- The Artist and Sector are notified of the communication call acknowledging receipt of the message.

- The Artist and Sector are notified of the firing.

- The unit sets its activity to firing and then "in position" at the end of the mission.

The behavior for the air defense battery is in response to the message "engage the target" which it receives from the regiment headquarters. The following actions are taken:

- The Artist and Sector are notified of the communication acknowledging receipt of the message.

- The Artist and Sector are notified of each time an air defense missile is fired.

- The unit sets its activity to "firing" during the mission and to "in position" after completion of the mission.

- The number of missiles which the unit fires is determined according to a Poisson distribution.

- A communication is made to the regiment headquarters on mission completion, and the Artist and Sector are notified of this this communication.

The other behavior for artillery batteries is in response to the message "fire preparatory fires" which it receives from Scheduler. The following actions are taken:

- The unit sets its activity to "firing".

- The unit notifies Artist and Sector of the firing.

- The unit sets it activity to "in position" at the mission completion.

3.2.14 Unit Property File

This file contains the generic unit property data, that is, each of the actors just above the instance level. There are forty generic units currently in the BEM. These include the combat, combat support and combat service support units at company level and their headquarters from battalion to division level. The parent of each of the action units, i.e., those at company level, is the generic actor Action-Unit. The parent of each of the control units, i.e., the headquarters units, is the generic actor Control-Unit.

Properties for the generic units include the parent unit, offspring units, numbers and types of vehicles, types of radios, radio networks used, times to clear from and close into position, convoy speed and convoy length. (See Appendix III.)

No actor behaviors are currently stored at this level.

3.2.15 Red Instance File

This file contains the instantiation of each of the 144 basic actors representing the Red units in the simulation. The actor parents are the generic units described in the above paragraph. The instance actors do not have offspring in the actor hierarchy. No behaviors are currently stored at this level.

It is at this level that the unit receives its individual properties. These include the starting and final location for the unit in the time slice; the unit's location in the military hierarchy to include superior unit, subordinate units, supporting artillery units, units supported, combat service support units and alternate headquarters. Additional properties include the radio frequencies on which the unit operates in order to communicate with superior, subordinate and supporting units.

3.2.16 Sensor

The generic object "Sensor" is the highest level object pertaining to sensors. It contains a property list as a template for lower-level sensor objects. The values of the elements of the property list are mostly unfilled at this level. The Sensor has only one behavior and that is related to movement. Its response to the message

"proceed to < position"

is to ask the Artist to move a sensor instance from its current position to the position specified and then to update the property "position" to the new location.

3.2.17 Sensor-Control-Unit

The generic object "Sensor-Control-Unit" contains the behavioral responses to the initial tasking of ground or air sensors. Thus, in response to a message

"send ground > type sensor to setup point > setup-pt
for > mission-duration minute mission"

a sensor control unit will find a ground sensor of the type required by checking through the list of sensors assigned to it until it finds one not currently tasked. Then a timing task will be constructed containing information related to the mission duration and duty cycle (if pertinent). A message will then be sent to the selected ground sensor ordering it to move to the specified setup point and to start its mission according to the constructed task. Had no ground sensor of the required type been available, the request would have been discarded with an appropriate response sent to the user via the text terminal of the BEM Controller Station.

For airborne sensors, the sensor, control unit would receive a message of the form

"send up airborne >type sensor in >leg-length Km
racetrack pattern > num-circuits times around with
initial pt >init-pt"

and search for and select the appropriate sensor in the way it did for ground sensors. The variable information passed in the message — the type of sensor, the leg length of the racetrack pattern, the number of circuits around the racetrack and the initial point of the racetrack — are sufficient to completely describe the activities of the sensor chosen. Again a message will be sent to the selected sensor passing timing and route information in a structured format.

A separate behavior exists for airborne IMINT (Photo) sensors since their behaviors are significantly different from other airborne sensors. Airborne IMINT sensors travel to specified points to take photographs instead of following a racetrack pattern. However, the behaviors for selecting a sensor, sending a sensor an order to move, and responding if no sensor is available are the same as for other sensors.

3.2.18 Sensor-Action-Unit

The Sensor-Action-Unit is a generic object which contains some of the behavioral responses to the messages sent by a sensor control unit. The behavior involved with moving an airborne IMINT sensor applies only to IMINT sensors and is therefore appropriately placed in the generic IMINT actor, one level down from the Sensor-Action-Unit.

Upon receipt of the ROSS message

"move ground sensor to >point with >task"

the Sensor-Action-Unit will ask the Artist to draw the sensor on the Ground Truth Display at its initial position. The Artist will also be asked to draw and erase the sensor's coverage area periodically according to its duty cycle. If the ground sensor is to be moved, a message will be sent to the Pathfinder actor asking it to construct for the sensor a route over the road network. The sensor will be placed on a list of "sensors-in-use" so that it cannot be retasked during its mission. If the sensor is an MTI, a message is sent to the Mathematician actor asking it to determine when the sector(s) are penetrated, i.e., when the sensor turns on and off. Information pertaining to the on/off status of the sensor will be sent to the Blue Sensor Control Station (BSCS) if it is turned on so that the appropriate sensor coverage may be displayed on the BSCS graphics terminal.

Another behavior in the Sensor-Action-Unit responds to the ROSS message

"move airborne sensor in racetrack pattern with > task
and > plan."

Again the Artist will be sent a message to display the airborne sensor at its initial position (always an airport). The Flightplanner actor will be asked to construct an airborne route for the sensor. If the sensor uses sectors directly to help with detection, Mathematician will be asked to determine when the sensor's coverage "penetrates" the relevant sector(s).The only other bet of the Sensor-Action-Unit responds to is the message

"fly to > position."

A sequence of these messages are created as a result of the route const by the Flightplanner and refined by the Mathematician. The response message principally involves updating current position of the Sens asking the Artist to display the movement of both the sensor and its re coverage area. Since airborne sensors turn their coverage off as they their racetrack pattern, the on and off status of the sensor will a reflected by those same messages to Artist. Finally, the current positi on/off status of the sensor will be sent to the Blue Sensor Control Stat that it may display the sensor's movement and coverage on its own gr terminal. Thus the sensor will be seen moving on both the Ground Tru BSCS displays.

3.2.19 Generic Sensors

There are five generic sensor actors — MTI, COMINT, CMCB, ELIN IMINT — all of which are created by the Sensor-Action-Unit. As previously, there are currently no formal sensor models in the BEM detections based on an event occurring within the geographic coverage a the sensor. When that detection occurs, a message is sent by the Sector (acting as an intermediary between threat and sensor units) to the involved asking it to

"check sensor detection of > unit."

Currently the sole response of all sensors is to consider the specified be detected upon receipt of this message and to cause a sensor report sent out to ANALYST or the Blue Sensor Control Station by sending a m to the BEM actor Interface. In a future version of the BEM, formal models will reside in the generic sensor actors and will make a determination based on terrain, signal attenuation, etc. as to wheth observable was actually detected.

The generic IMINT actor, however, has other behaviors not related to sensor detection but to sensor movement. These are behaviors associated with the "move" and "fly to" messages first discussed in the previous section on the Sensor-Action-Unit. Although the movement of airborne IMINT sensors is different from the racetrack patterns of other airborne sensors, the response to these messages is functionally equivalent to the corresponding behaviors in the Sensor-Action-Unit.

3.2.20 Sensor Instance File

The sensor instance file contains the ROSS instances for the sensor control unit and all sensors used in the BEM simulation. There are no behaviors in this file, only properties relating to sensor range, coverage, speed, type, and beginning and ending points. This file contains all sensors known to the BEM although they need not all be tasked by the Scenario actor during the running of a simulation. By keeping a group of sensors in reserve, there are always some available for dynamic tasking from the Blue Sensor Control Station.

3.3 Considerations For Using ROSS/LISP

The use of the ROSS programming language is a very positive feature of the current BEM. Even though a message-passing environment is radically different from the standard procedural environment, acclimation to it is fast and easy. The use of object-oriented programming techniques is most advantageous at the command and control level. At this high level, the ROSS messages and objects closely conform to actual communications between real-world actors. This is true for both threat units and sensors. However, at a somewhat lower level, where auxiliary actors and their messages have perhaps no real-world counterparts, the inherent modularity of ROSS code can still enhance the intelligibility of many processes. It is at the very low levels, usually involving computational processes, that the efficacy of message passing breaks down. There is an overhead attached to processing messages; therefore, the point at which message passing no longer clarifies a process is the point at which it should be abandoned in favor of LISP function calls.

Determining where this point is, however, is largely judgmental, and can even be a matter of style. Many computational processes which might otherwise have been placed in the Mathematician exist instead in pure LISP functions. Placing a computation in the Mathematician means that it will be performed as a result of a message sent instead of a LISP function call. The decision as to which processes to single out in this manner is largely judgmental and related to a sense of aesthetics and whether the implementor thinks that an English-worded ROSS message adds something to the understanding of the simulation.

Another consideration in the use of ROSS/LISP is the resulting development and demonstration environments. For small programs, use of the interpreter is a productive method for constructing and debugging. The process of compiling ROSS code is not fast and the compiler still has some vagaries attached to its use. However, as the BEM has grown larger, it has been necessary to run a good portion of the code compiled even when debugging. For demonstrations, even though almost all code is compiled, the BEM simulation may run at best with a simulation to real time ratio of 2:1 or 3:1. For many uses this is not undesirable, but as the BEM continues to grow, speeds less than this would be unacceptable.

4.0 FUTURE ENHANCEMENTS OF THE BEM

Work has already begun on what is tentatively called BEM II. Whereas the current BEM models the activities of a Soviet division with company-level resolution, BEM II will model echelons above corps with battalion-level resolution. Some other areas of future development which will be discussed in the following sections are:

- o formal sensor models
- o two-sided simulation
- o speed-up of processing.

4.1 Formal Sensor Models

As was emphasized in the previous section on Generic Sensors (Section 3.2.15), the BEM models sensor detection of threat units by determining whether a unit or unit activity lies within a relevant sensor's coverage area. This is essentially a first step in the total process of detecting units. The second step involves the use of formal sensor models which are not now present in the BEM. In the future, detection "candidates" from the first step may be presented to these formal sensor models which use additional criteria to determine whether or not a detection has occurred. Factors such as terrain, signal attenuation, weather, and sensor errors can all be considered in the design of formal sensor models. In addition, various approaches to modeling multiple sensors on a single platform will be considered. More detailed and realistic modeling of sensors must be considered in a future BEM if it is to become a useful testbed for sensor deployment studies.

4.2 Two-Sided Simulation

The current BEM is one-sided, i.e., there is no modeling of Blue (NATO) forces other than sensors. Consequently, somewhat artificial means (random generators) are used to cause Soviet radar emissions and artillery firings, etc. In a two-sided model, these events would most likely stem from a reaction to Blue force actions. It is therefore desirable to create a representation of the Blue force side (other than sensors) so that the BEM can further serve as a testbed for command and control studies.

4.3 Speed-Up Of Processing

The current speed limitations of the BEM were discussed in Section 3.3. The BEM simulation runs slowest, of course, when all three components of a demonstration are active — the ANALYST, the Blue Sensor Control Station, and the BEM proper. The ANALYST currently runs during a demonstration on the VAX 11/780. When the VAX and the SPL's LISP machine are linked together, the ANALYST will run on the LISP machine, thus removing a significant computational burden from the VAX. The Blue Sensor Control Station can relieve some of the computational burden by not only undergoing a careful adjustment of its priorities, but also a restructuring of its menuing to a non-cursor mode. In the BEM proper, many computations regarding threat movement are currently preprocessed since that movement is determined from the SCORES data. Some subsequent computations related to this movement are currently performed during the simulation, but could also be preprocessed. With these changes and a more careful look at ways to optimize the modeling effort, the BEM II should be able to maintain an acceptable running time.

APPENDIX I

SENSOR RESEARCH FOR THE BEM

TABLE OF CONTENTS

1.0 INTRODUCTION

The purpose of this appendix is to describe the research that has been done so far on U.S. sensors for use in the Battlefield Environment Model (BEM). This research covered two aspects of sensors - employment methods and properties; the generic sensor types discussed are SIGINT (COMINT and ELINT), MTI, CM/CB, and IMINT; jammers are not included now, although they will be in the future.

The research was conducted mainly by consulting personnel at the Signals Warfare Lab (SWL) [1], the MITRE Corporation[2], and DIA[3], as well as the referenced documents.

2.0 SENSOR DATA

This section begins with a brief discussion of the employment of each generic sensor type which is now modeled in the BEM or is intended to be modeled in the near future; this discussion is followed by a listing of the properties for each type, in chart form. The information is what was necessary to support the determination of time-and-space overlaps of sensors and units used in the initial modeling effort; see Section 3.1 for more extensive sensor parameters to be used later in the formal sensor models.

2.1 Sensor Employment

2.1.1 SIGINT Employment

Difficulties in modeling IEW are caused by the fact that our current IEW technology has not been tested in actual combat; therefore there are no firm and precise procedural guidelines for performing IEW functions. What follows is a general description of those aspects of SIGINT sensor employment to be modeled in the BEM and our current understanding of the procedures that will ultimately be used.

For our purposes, SIGINT (Signals Intelligence) will be considered to be performed by COMINT (Communications Intelligence) and ELINT (Electronic Intelligence) sensors. COMINT sensors are intended to intercept and locate emissions from enemy radios (current model requirements deal only with COMINT externals); ELINT sensors perform a similar function on enemy radar emissions.

COMINT and ELINT sensors are combined here because they involve similar employment techniques; however, their properties differ enough to warrant separate representation in the model.

For purposes of discussion, the basic functions of COMINT and ELINT sensors are divided into three processes — tasking, collection, and reporting.

Tasking

Mission tasking for these sensors comes from the S2/TCAE at the CEWI Group (corps echelon) or CEWI BN (division echelon). Tasking can include:

- mission start and end time
- location of sensors - For airborne sensors this could be instructions to proceed to point A and begin flying an ellipse around points A and B.
- targets - This may be a list of frequencies to scan a specified target type, such as "all artillery battalion command posts," or instructions to collect against any emitters for which the sensor has the capability in a given geographical area.

Collection

When tasking is received, the sensors will move into position and begin collecting. They will be shown moving within their respective areas — division for rotary wing and some ground sensors, corps for fixed-wing and some ground sensors.

Since COMINT and ELINT sensors are passive and not easily subject to detection by the enemy, the sensors can be on for the duration of the mission, less travel time to and from position. However, any data links from the sensor platform are detectable, and are always on when there is no operator in the aircraft. When there is an operator in the aircraft, he can report signals as they are intercepted or wait until after the mission; therefore, the data link may be on or off at any given moment.

The sensors usually operate in groups of two or three for DF fixing; lines of bearing from individual sensors are correlated to produce target locations. The geographical area covered is dependent on the sensor's sector scan (in degrees) and height (for airborne sensors).Movement of airborne sensors is confined to the mission itself, e.g., an elliptical flight path behind the FEBA, and travel to and from that path; movement of ground sensors is based on the combat situation and is generally determined by the movement of the supported echelon (about twice a day for division).

Reporting

Three types of reports are sent back to the TCAE:

1) sensor location, operational status

2) technical reports - These may include frequencies intercepted and lines of bearing for analysis, and will be passed on to the user after analysis, stripped of any codeword information.

3) user product (TACREP, TACELINT) - This is immediately usable information normally requiring no analysis. Usually it goes through the TCAE first, but may be sent directly to the ultimate user if necessary and if it contains no codeword information.

2.1.2 Moving Target Indicator (MTI) Employment

The BEM represents both ground-based and airborne MTI radars, the ground based being a generic ground surveillance radar (GSR) and the airborne being a side-looking airborne radar (SLAR).

2.1.2.1 Ground--Based MTI (GSR)

Tasking

The GSRs with which the BEM is concerned are organic to the Ground Surveillance Company of the CEWI Battalion, although they are usually attached to maneuver brigades or battalions. Therefore, depending on the sensor used, the radar teams may be tasked by maneuver battalions, brigades, or divisions.

Collection

GSR's are placed in stationary positions, usually 500 meters to 1 Km behind the FEBA and, depending on the sensor, may be carried by personnel or on vehicles. They are used to detect moving targets, either personnel or vehicles, and employ random operating periods of short duration to avoid detection.

Reporting

GSR's provide information on the range, azimuth, elevation, direction of movement, and vehicle type (tracked or wheeled); since GSRs are usually attached to a maneuver echelon, reports are usually sent to the intelligence organization of that supported echelon.

2.1.2.2 Airborne MTI (SLAR)

Tasking

Various platforms such as the Army's OV-1D and Air Force high-altitude aircraft may be furnished with SLAR equipment; the platform used so far in the BEM is the OV-1D fixed-wing aircraft. Since this aircraft is a corps asset,tasking is done by corps through the corps mission management element. The SLAR is tasked to detect enemy force movements and changes in enemy activity patterns; it may also perform rear area protection (RAP) and operations security (OPSEC) support function.

Collection

The SLAR is used mainly as an MTI collector but also has a fixed target indicator (FTI) capability: its output is a two-channel film, one channel showing fixed-target imagery and the other showing bright spots indicating movement. The platform generally flies a racetrack pattern across the corps front, approximately 30 Km behind the forward edge of the battle area (FEBA). The SLAR is unable to distinguish target types or to differentiate enemy and friendly targets; it reports only the radial components of the target's velocity, not the transverse.

Reporting

The SLAR film is processed in flight, with a near real-time display in the cockpit; in addition, the output may be data linked for simultaneous display at ground data terminals located with the supported units. The data is forwarded to the corps CEWI Group (TCAE).

2.1.3 CM/CB Employment Tasking

The CM/CB radars are organic to the division target acquisition battery, and are tasked by division artillery.

Collection

CM/CB radars are used to locate enemy mortars, artillery, and rockets. They are deployed several kilometers behind the FEBA and operate by back-plotting the weapon's trajectory.

Reporting

Target location and time are data-linked to the direct support or general support artillery battalions.

2.1.4 IMINT Employment

The two types of imagery researched for in the BEM are photography and imaging radars; representation of infra-red (IR) photography is not planned for the near future. Imagery as currently represented in the BEM consists of OV-1D overflight; the following paragraphs on photography and imaging radars describe future capabilities of the BEM.

2.1.4.1 Photography

Tasking

Because the immediate need is for photography deeper in the enemy's rear areas, the BEM will be representing penetration missions rather than stand-off photography. The major platform for this type mission would be an Air Force high-altitude aircraft; therefore, the tasking for the sorties is an Air Force responsibility, with a specified number of sorties allotted to fill Army photo requirements.

Collection

Photo missions will be flown once or twice a day, usually between 1000 and 1400 hrs when lighting conditions are favorable. The flight path used in the BEM will be a straight line, since the high speed of the aircraft allows it to cover the BEM's battlefield in a matter of seconds and the aircraft requires

200 miles for turning. The aircraft cameras take a series of still pictures of an area approximately 40 miles wide.

Reporting

The film must be taken to a ground imagery interpretation center at corps for developing and interpretation; this can cause a delay of four to five hours between collection and reporting to the corps CEWI Group (TCAE).

2.1.4.2 Imaging Radars

The imaging radar represented in the BEM is the Synthetic Aperture Radar (SAR).

Tasking

Tasking of the SAR depends on the platform used. The platforms considered here are the Army's OV-1D aircraft and an Air Force high-altitude aircraft (see paragraph 2.1.2.2 for OV-1D tasking and 2.1.4.1 for Air Force high altitude aircraft tasking).

Collection

Collection is usually done during penetration missions and results in a precise digital image that has the appearance of a photographic negative.

Reporting

Since the output is a digital image, there is no delay for ground processing; the image can be linked directly to the CEWI Group at Corps (TCAE).

2.2 Sensor Properties

Table I-1 below shows the properties currently used in the BEM for each generic sensor type modeled; see Section 3 for more detailed properties to be used in the formal sensor models.

TABLE I-1
SENSOR PROPERTIES

	GROUND		FIXED-WING AIR			ROTARY-WING AIR		
	SECTOR SCAN	RANGE	SECTOR SCAN	RANGE	PLATFORM SPEED	SECTOR SCAN	RANGE	PLATFORM SPEED
COMINT	360°	30KM-VHF 1000 + KM-UHF	360°	320KM at 20,000 ft (alt = 10,000 -20,000 ft)	160Kn/hr	360°	90KM-VHF 1000+KM-UHF (at 3000 ft)	90 Kn/hr
ELINT	90°	30KM	70°	225KM at 10,000 ft (25,000= max alt 10,000 = normal)	180Kn/hr			
MTI	*180-1955 mils	*3,000-18,000m * depending on sensor chosen	SLAR 120°	100KM at 7500 ft	180Kn/hr			
CM/CB	60°	15-24KM (CM) 30-50KM (CB)						
IMINT (PHOTO)			penetration mission 40 mile wide area covers battlefield in 3 seconds	200-300 miles per mission	2,000 Kn/hr			
(SAR)					180Kn/hr (on OV-1D)			

3.0 FUTURE EFFORTS

Future efforts of concern here are in two areas - the addition of formal sensor models and the use of realistic message formats for sensor reports. The paragraphs below outline the data required for these efforts.

3.1 Data for Formal Sensor Models

Listed in the tables below are suggested properties of each of the sensor types which will be represented in the future by a formal sensor model. For each sensor type, properties are given for the platform, collector, and ground processing station; some of these properties were suggested by references 2,7, and 8. Properties are listed for jamming systems and remotely monitored sensors (REMS), although these systems will probably not be included in the near future.

It should be understood that these properties are likely to change somewhat; they are included here as an indication of the level of detail planned.

3.2 Message Formats

THE COMINT reports in the BEM currently contain only message internals - such characteristics as frequency and start/stop times. It is expected that in the future, report representation will be augmented to include the text of COMINT interceptions as well as formatted messages such as TACELINT, PEAR, SITREP, etc.

Table I-2
Generic SIGINT System
Platform Characteristics

Factor	Units
Payload	lb.
Range	km
Time on station	minutes
Max speed	knots or km/h
Operational speed	knots or km/h
Max height	meters
Operational height	meters
Fuel load	lbs.
Crew required	n
Survivability index	n
Tracks-Ground system only	n
Wheels-Ground system only	n
Type	ground or air
Location	km from FEBA, or x, y, (z)
No. of platforms	n
Vulnerabilities	ADA, weather, etc.
Mean time between failures	hours
Mean time to repair	hours

Table I-3
Generic SIGINT System
Collector Characteristics

Factor	Units
Crew required	n
Frequency range	n .. nMHz
Modulation	AM, CW, SSB, PM (COMINT only)
Sensitivity	db
Location error - range	meters
Location error - azimuth	degrees
DF fix accuracy	meters
Coverage range	km
Sector scan	degrees
Revisit time	seconds
Mean processing time	seconds
Saturation rate	# intercepts/minute
Method of reporting	means
Reporting speed	# reports/minute
Freq. measurement error	MHz
PRF measurement error	sec. (ELINT only)
PW measurement error	sec. (ELINT only)
Frequency hop capability	Y/N (ELINT only)
Pulse stagger capability	Y/N (ELINT only)
Constraints	LOS, distance, etc.
Data link	Y/N, to where
Mean time between failures	hours
Mean time to repair	hours

Table I-4
Generic SIGINT System
Ground Processing Station Characteristics

Factor	Units
Crew required	n
Tether range to platform	km
# collectors at one time	n
Mean processing delay	minutes
Mean communications delay	minutes
Saturation rate	# reports/minute
Location	km from FEBA, or x, y
Number of receivers	n
Channels per receiver	n
Collectible frequency (each channel)	n..nMHz

Table I-5
Generic Photo Imagery System
Platform Characteristics

Factor	Units
Range	km
Time on station	min.
Max speed	knots
Operational speed	knots
Max height	meters
Operational height	meters
Fuel load	lb.
Crew required	n
Survivability index	n
Number of platforma	n
Location	x, y, z
Vulnerabilities	ADA, weather, etc.
Mean time between failures	hours
Mean time to repair	hours

Table I-6
Generic Photo Imagery System
Collector Characteristics

Factor	Units
Mean process delay	seconds
Data linked	n
- obscuration factors	foliage, clouds, light conditions
- min/max look angle	degrees
- area covered/opn.ht.	sq.km
Crew Required	n
Collection means	film or electro-optical (if electro-optical, ground station processing delay is 0)
Mean time between failures	hours
Mean time to repair	hours

Table I-7
Generic Photo Imagery System
Ground Processing Station Chacteristics

Factor	Units
Mean process delay	min.
Mean commo delay	min.
Crew required	n
Saturation rate	reports/min.
# collectors at a time	n
Location	km from FEBA,or x, y

Table I-8
Generic IR System
Platform Characteristics

Factor	Units
Range	km
Time on station	min.
Max speed	knots
Operational speed	knots
Max height	meters
Operational height	meters
Fuel load	lbs.
Crew required	n
Survivability index	n
Number of platforms	n
Location	x, y, z
Vulnerabilities	ADA (because of low flight), weather, etc.
Mean time between failures	hours
Mean time to repair	hours

[illegible]

[illegible]	Units
[illegible]	[illegible] S
[illegible]	[illegible] S, [illegible] where
[illegible]	[illegible]
[illegible]	temperature, emissivity of objects
[illegible]	degrees
[illegible]	[illegible]
[illegible]	[illegible]
[illegible]	[illegible]
[illegible]	[illegible]
[illegible]	[illegible] photo [illegible], requires low flight

[illegible]

[illegible]	Units
[illegible]	[illegible]
[illegible]	[illegible]
[illegible]	[illegible]
[illegible]	[illegible] min.
[illegible]	[illegible]
[illegible]	[illegible] FEBA, or x,y

Table I-11
Generic Imaging Radar System
Platform Characteristics

Factor	Units
Type	air, ground, manpack
Range	km
Time on station	min.
Max speed	knots
Operational speed	knots
Max height	meters
Operational height	meters
Fuel load	lb.
Crew required	n
Survivability index	n
Number of platforms	n
Location	x, y, (z)
Vulnerabilities	ADA, weather, etc.
Mean time between failures	hours
Mean time to repair	hours

Table I-12
Generic Imaging Radar System
Collector Characteristics

Factor	Units
Type	radar type
Crew required	n
Area covered at operational height	square km
Mean process delay	seconds
Data linked	Y/N, to where
Range at operational height	meters
Angular resolution	degrees
Output	digital display, film
Vulnerabilities	active emitter
Mean time between failures	hours
Mean time to repair	hours

Table I-13
Generic Imaging Radar System
Ground Processing Station Characteristics

Factor	Units
Mean processing delay	min.
Mean commo delay	min.
Crew required	n
Saturation rate	reports/min.
# collectors at a time	n
Location	km from FEBA, or x, y

Table I-14
Generic MTI System
Platform Characteristics

Factor	Units
Type	air, ground vehicle, manpack
Location	x, y, (z)
Number of platforms	n
Range	km
Time on station	min.
Max speed	knots or Km/h
Operational speed	knots (Air only) or Km/h
Max height	meters
Operational height	meters
Fuel load	lbs.
Crew required	n
Survivability index	n
Vulnerabilities	ADA, weather for air
Tracks	n, ground only
Wheels	n, ground only
Mean time between failures	hours
Mean time to repair	hours

Table I-15
Generic MTI System
Collector Characteristics

Factor	Units
Target types detected (tracked, wheeled)	personnel, vehicles
Constraints	LOS, etc.
Data linked	Y/N, to where
Velocity threshold/target type	km/n
Crew required	n
Resolution	meters
Mean process delay	seconds
Range resolution	meters
Angular resolution	degrees
Output	digital display, film
Vulnerabilities	active emitter
Mean time between failures	hours
Mean time to repair	hours
Power	Watts
Crew required	n
# collectors at one time	n
Mean processing delay	min.
Mean commo delay	min.
Saturation rate	# reports/min.
Location	km from FEBA, or x, y

Table I-16
Generic CM/CB System
Platform Characteristics

Factor	Units
Number of platforms	n
Location	x, y
Fuel load	lb.
Crew required	n
Survivability index	n
Vulnerabilities	active emitter
Tracks	n
Wheels	n
Mean time between failures	hours
Mean time to repair	hours

Table I-17
Generic CM/CB System
Collector Characteristics

Factor	Units
Frequency range	n..n
Range resolution	meters
Angular resolution	degrees
Reporting speed	reports per minute
Constraints	limited operator time
Output	real-time target location; hard copy
Mean processing delay	seconds
Target types detected	artillery, mortars
Data linked	Y/N, to where
Vulnerabilities	active emitter
Crew required	n
Mean time between failures	hours
Mean time to repair	hours

Table I-18
Generic CM/CB System
Ground Processing Station
Characteristics

Factor	Units
Crew required	n
# collectors at one time	n
Mean processing delay	min.
Mean commo delay	min.
Saturation rate	# reports/min.
Location	km from FEBA, or x, y

Table I-19
Generic REMS System
Collector Characteristics

Factor	Units
Sensing life	hours
Frequency range	n..nMHz
Accuracy	emplacement
Data link	Y/N, to where
Range	km
Constraints	emplacement may require penetration of enemy territory
Vulnerabilites	environmental noise, heat vibrations, etc.
Mean processing time	seconds
Location	x, y
Output	target locations, number, direction
Saturation rate	# reports/min.

Table I-20
Generic REMS System
Monitor Characteristics

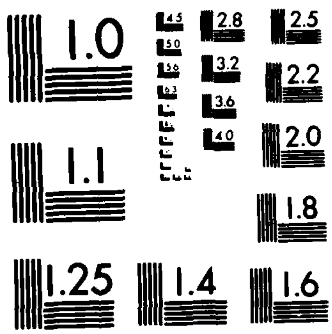
Factor	Units
Channels	n
# collectors monitored	n
Location	x, y; or with what unit
Mean processing delay	min.
Mean commo delay	min.
Saturation rate	# reports/min.

Table I-21
Generic Jamming System
Platform Characteristics

Factor	Unit
Type	air or ground
Crew required	n
Fuel load	lbs.
Location	x, y
Survivability index	n
Vulnerabilities	AD for air
Tracks	n
Wheels	n
Mean time between failures	hours
Mean time to repair	hours
Max speed	knots or km/h
Operational speed	knots or km/h

AD-A143 753 THE BATTLEFIELD ENVIRONMENT MODEL (BEM)(U) MITRE CORP MCLEAN VA MITRE C3I DIV R S CONKER ET AL. SEP 83 MTR-83W00245 F19628-84-C-0001 2/3

UNCLASSIFIED F/G 15/7 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Table I-22
Generic Jamming System
Jammer Characteristics

Factor	Units
Frequency range	n..n
Power	Watts
Data link	Y/N, to where
Constraints	frequency limits to avoid jamming friendly emitters
Vulnerabilities	enemy ECCM
Location	x, y
Output	FM, CW, AM, SSB, PM
Mean time between failures	hours
Mean time to repair	hours

REFERENCES

1. Department of the Army, U.S. Army Field Manual 34-10, Combat Electronic Warfare Intelligence Battalion (Division) (Coordination Draft), June 1979.

2. The MITRE Corporation, Army Intelligence Capabilities Reference Handbook (U), MTR-79W00461, R. P. Bonasso, November 1979, SECRET.

3. U.S. Army Command and General Staff College, Electronic Warfare Operations, July 1980.

4. U.S. Army Electronics Materiel Readiness Activity, USAEMRA Tactical SIGINT/EW Reference Guide, July 1982. 5.Discussions with MITRE personnel.

5. Discussions at Signals Warfare Lab, May 5, 1983.

6. Discussions with MITRE personnel.

7. The MITRE Corporation, Current, Planned, and Future Army Sensor Capabilities - Working Input to Critical Nodes Targeting Study)U), DRAFT, E. J. Boyle, B. Hart, June 1982, SECRET.

8. TRW, POSSIM Methodology Manual (Preliminary), (U), CDRL A018, October 1979, SECRET

9. Department of the Army, Training Circular 34-50, Reconnaissance and Surveillance Handbook, January, 1980

APPENDIX II

THREAT RESEARCH FOR THE BEM

TABLE OF CONTENTS

1.0 INTRODUCTION

1.1 Purpose

This appendix is intended to describe the research conducted to derive the Soviet threat data used to support the Battlefield Environment Module (BEM) developed in MITRE's Secure Processing Laboratory (SPL). The BEM supplies the battlefield environment — the Red units, their moving/shooting/emitting events, and Blue sensors — and produces threat observables and Blue sensor detection for use by ANALYST, a developing expert system for intelligence fusion.

1.2 Outline of the Appendix

Section 2 provides a short summary of the intent of the threat research discussed in this appendix and some of the problems encountered. Section 3 describes the scenario used as a basis for the SPL simulation and the development of that scenario over time. Section 4 describes the representative force structure used, and Section 5 details the characteristics of the generic unit types used in the simulation. Finally, Section 6 provides a glimpse of the future efforts needed in threat research for the SPL. Appendix III of this paper, published separately, consists of the data lists, such as tables of unit positions, radio nets, and generic unit data.

2.0 SUMMARY

The picture that emerges from this research is intended to fit the purposes of the SPL; that is, it is required to support a scenario that will produce a sufficient number of threat observables with sufficient realism to adequately test the fusion capability of the ANALYST system. For this reason, it is intended to be a representative portrayal of the threat rather than an exhaustive one.

In researching a subject such as the Soviet threat, several factors conspire against a perfect representation: threat organizations and equipment are constantly changing while the system needs to be continuously in use and development, and various sources provide conflicting data which need to be reconciled before being entered into the system. In addition, the classification of sources can be a problem; because of computer facility restrictions, it was necessary in the earlier stages of research to keep the data at the unclassified level. It is now possible to run SECRET level data, with a higher level capability planned for the future. These increased capabilities will necessitate changes in the data; it is expected that the object-oriented approach used in the evolving system will facilitate incorporating those changes into the system.

3.0 SCENARIO

3.1 *Description and Sources*

The scenario used for the simulation is based on the Scenario Oriented Recurring Evaluation System (SCORES), Europe I Sequence 2A, in which the U.S. Army V Corps defends/delays against an attack by the Soviet 1st Guards Tank Army east of Fulda, in the Federal Republic of Germany. In keeping with the objective of producing threat moving, shooting, and emitting observables, only Soviet units are portrayed in the simulation. In order to provide a representative sampling of units and procedures, the organizations selected for the simulation are a first-echelon tank division, the 6th Guards Tank Division, a second-echelon motorized rifle division, the 27th Motorized Rifle Division, and essential Army-level units. The time period covered is from 0330 on the second day of combat to 0330 on the third day.

The sources used as a starting point for the scenario development were the SCORES documentation and MITRE documents.[1-7] Alterations made to the scenario portrayed by these documents are described in Section 3.3. In addition, some unit capabilities were referenced from documentation for the Corps/Division Evaluation Model (CORDIVEM).[8]

3.2 Unit Representation

Each unit in the scenario is given a unit type number from one to sixty; this number corresponds to one of the unit types in the Generic Unit Data File and indicates the equipment and other characteristics of the unit. The unit is further identified by a unique, eight-digit unit identification number of the form CC/BB/RR/DD, which identifies the company number and the battalion, regiment, and division to which the unit belongs. Segments of the format which do not apply are filled with zeros; for example, the antitank battalion command post which is organic to Army has the number 00/83/00/00, since it is not a company and is subordinate to neither a regiment nor a division.

Each unit is assigned a geographical position for the beginning of each timeslice of the scenario period; these timeslices are 0330—0930 D+1, 0930—2130 D+1, 2130 D+1-0330 D+2, and 0330—0930 D+2. For each of these timeslices, each unit is given a status code indicating "assembled," "moving," or "in contact," according to its activity in that time period; if the unit is an artillery unit, it is also given a number indicating the regiment it supports during the timeslice.

Below is a listing of the unit types and their basic capabilities, organized by functional area.

3.3 Scenario Development

Several changes were made to the scenario as portrayed in SCORES. First, the level of resolution was extended downward from the battalion to the company. Next, unit positions were refined to more closely reflect terrain restrictions. Then units were added to the scenario which had not been included in SCORES, such as division and maneuver regiment forward command posts, and the antitank battalion and independent tank battalion of the second echelon division. Finally, the representation was expanded upward to include Army-level units.

The system is currently running at the level of one division, with data available for the two divisions and Army units.

Table II-1
MANEUVER UNITS

TYPE	NAME	CAPABILITIES
1	MR CO	maneuver, delivery of fires
2	NOT USED	
3	MR BN CP	C^2 of 3 MR CO's
4	MR RGT FWD CP (forward position of CMDR)	C^2 of 3 MR BN's & 1 TK BN
5	MR RGT MAIN CP	C^2 of 3 MR BN's & 1 TK BN
6	MR DIV FWD CP 1 Independent TK BN (commander's forward position)	C^2 of 3 MR RGT'S, 1 TK RGT,
7	MR DIV MAIN CP 1 Independent TK BN	C^2 of 3 MR RGT's, 1 TK RGT,
8	TK CO	Maneuver, delivery of fires
9	NOT USED	
10	TK BN CP	C^2 of 3 tank companies
11	TK RGT FWD CP Commander	Forward position of Tank RGT
12	TK RGT MAIN CP	C^2 of 3 tank BN'S
13	TK DIV FWD CP Commander	Forward position of Tank Div.
14	TK DIV MAIN CP	C^2 of 3 Tank RGT'S & 1 MR RGT
44	TK ARMY FWD CP Commander	Forward position of Tank Army
45	TK ARMY MAIN CP	C^2 of Tank Division & 1 MR Div.

Table II-2
ENGINEER UNITS

TYPE	NAME	CAPABILITIES
52	Army Eng. RGT CP	C^2 of Army Construction BN (type 53) & Army Engr. BN (type 33)
53	Army Construction BN (considered action unit, not broken further)	o Emplace and remove span bridges o Road construction & maintenance o Operate Sawmill
33	Eng. BN (Army) (considered action unit, not broken further)	o water crossing-boats, ferries, amphibians, truck launched bridges o Emplace & remove counter mobility obstacles, (mines, abatis, booby traps, craters, etc.)
56	Army Pontoon RGT (action unit, not broken down)	o Emplace & remove pontoon bridges o Launch and recover amphibians
57	Army Assault Crossing BN (not broken down	o Launch tracked ferries and amphibians (not bridges)
33	Eng. BN CP (Div)	C^2 of Pontoon Co; Assault Crossing Co, (also, there is a "Road-Bridge Const. Co." at Div., it is omitted along with "technical Co.", etc.)

----

* (Capabilities are based on inferences from equipment tables, and some capabilities listed in Red FARO'S; however, Red FARO's show different units, so again, inferences were made.)

Table II-2
(Concluded)

TYPE	NAME	CAPABILITIES
37	Pontoon Co. of Div Eng. BN	See #56
38	Assault Crossing Co. of Div. Eng. BN	See #57
28	Eng. Co. of Maneuver Rgts.	See #33 (Eng. BN, Army)

Table II-3
ARTILLERY UNITS

TYPE	NAME	CAPABILITIES
15	ARTY BTRY (122)	fire support to maneuver regiments (Div. artillery RGT)
16	ARTY BN CP (122)	C^2 of 3 - 122 Batteries (Div. Artillery RGT)
17	GUN-HOW or Howitzer BTRY (152)	fire support to division units (Army Artillery RGT)
18	GUN-HOW or Howitzer BN CP (152)	C^2 of 3 - 152 Batteries (Army Artillery RGT)
19	GUN BTRY (130)	fire support to division units (Army Artillery RGT)
20	GUN BN CP (130)	C^2 of 3 - 130 Batteries (Army Artillery RGT)
21	ARTY RGT CP (Div. or Army)	C^2 of artillery batteries for Division or Army
41	TGT. ACQ. BTRY (ARTY RGT., Div. or Army)	Radar acquisition of ground targets
22	FROG BTRY	Chemical, nuclear coventional, long range fire capability against targets in enemy area
23	FROG BN CP	C^2 of 3 FROG batteries
24	MRL BTRY	Multi-barreled rocket launchers, heavy fire on high-priority targets at decisive points, chemical or conventional
25	MRL BN CP	C^2 of 3 MRL batteries
42	NOT USED	
43	NOT USED	

(Table II-3)
(Concluded)

TYPE	NAME	CAPABILITIES
49	SCUD Bde CP	C^2 of 3 SCUD BN'S
50	SCUD BN CP	C^2 of 3 SCUD Btry's
51	SCUD Btry	Fire support for Army operations (conventional, nuclear, or chemical missiles to range of 280 KM, tracked or wheeled vehicle launcher)
58	Antitank BN Army/MR Div	C^2 of 2 Antitank Batteries and 1 ATGM Battery
59	Antitank Btry. Army/MR Div	Provide antitank support to unit (guns)
60	*ATGM Btry Army/MR Div	Provide antitank support to maneuver units (guided missiles) - effective range 4-5 Km

----

*Anti-tank missile battery also exists in each TK & MR RGT, but has not been included here.

Table II-4
AIR DEFENSE UNITS

TYPE	NAME	CAPABILITIES
26	SA-6 BTRY	Engage low-to-medium altitude attacking aircraft and helicopters; area defense to division, point defense to maneuver regiments, Division HQ,and FROG BN
39	SA-6-TGT-ACQ BTRY	Radar acquisition of enemy aircraft targets
27	SA-6 RGT CP	C^2 of SA-6 BTRY's, & TGT ACQ BTRY
46	SA-4 Bde CP	C^2 of 3 SA-4 BN'S
47	SA-4 BN CP	C^2 of 3 SA-4 Btry's
48	SA-4 Btry	Acquire & engage medium-to-high altitude aircraft attacking divisions, Army HQ, or SSM Bde (No TAB for SA-4; acquisition radars are in the batteries)

Table II-5
CSS UNITS

TYPE	NAME	CAPABILITIES
29	MTCE CO. (of maneuver RGT's)	Light repairs, clear damaged vehicles from roads, report to Div. MTCE BN repairs beyond capabilities
34	Div. MTCE BN	Division level repair, recovery, clearing routes of damaged vehicles
30	Medical Co. (of RGT's)	Emergency medical treatment, evacuation to Regimental Medical Points from BN Medical Points, notify Div. Med BN to pick up casualties
35	Div. Med. BN	Collection from Regimental Medical Points to Divisional Reception Stations, triage, treatment (including surgery) evacuation to Army-Front hospitals
31	*Motor Transp. Co. maneuver RGT's	Transport supplies to maneuver BN'S (all classes of supply)
32	*Supply & Service Pltn (of Maneuver RGT'S)	Food and water supply
36	*Motor Transp. BN (Div or Army)	Transport supplies to maneuver regiments (or divisions)("all classes, but apparently not POL at Army since there is an Army POL BN)

TABLE II-5
(Concluded)

TYPE	NAME	CAPABILITIES
54	Army Motor Transp RGT CP	C^2 of Army Motor Transp. BN Army POL BN
55	*Army POL Transp. BN CP	POL transp. down to Div

---

*In maneuver RGT'S, MT CO'S are assumed to do POL supply (since they have the POL trucks) along with ammunitition; the S&S CO'S are assumed to do food-water (since they have kitchen and water trailers). At division, MT BN does all 3 (with AMMO CO, POL CO'S & S&S Pltn).

4.0 FORCE STRUCTURE

Because of the equipment first used, the scope of the force representation was initially restricted to one first-echelon tank division at company-level resolution. The data has since been expanded to include a second-echelon motorized rifle division as well as tank Army-level units. Only ground forces are considered.

Figure 1-1, 1-2, and 1-3 show the current scope of the force structure. This force structure, while representative of the major capabilities of the Soviet force at this level,[9,10,11] is not an exact replica of it. Since the research was conducted over a three-year period, it is likely that force structure changes have occurred which have not been incorporated yet; the known omissions are discussed below.

The Soviet Army now includes a motorized rifle battalion in the tank regiment; this unit should be added. The signal companies of the tank and motorized rifle regiments were not included since our main concern was the existence of communications traffic for Blue sensors to pick up, not the establishment and maintenance of communications facilities; however, when the simulation becomes two-sided these units should be modeled as significant actors. The same is true of the reconnaissance companies and chemical defense companies. The anti-aircraft artillery and missile batteries of the motorized rifle regiments are not explicitly modeled, their weapons and radars being included in their respective regimental command posts. The antitank missile battery of the motorized rifle regiment should be included in the future; however, a division-level antitank capability is represented now by the antitank missile battalion of the motorized rifle division. The mortar platoon of the motorized rifle regiment was not included because at the time the research was conducted, no Blue counter-mortar radars were modeled; this platoon should be included.

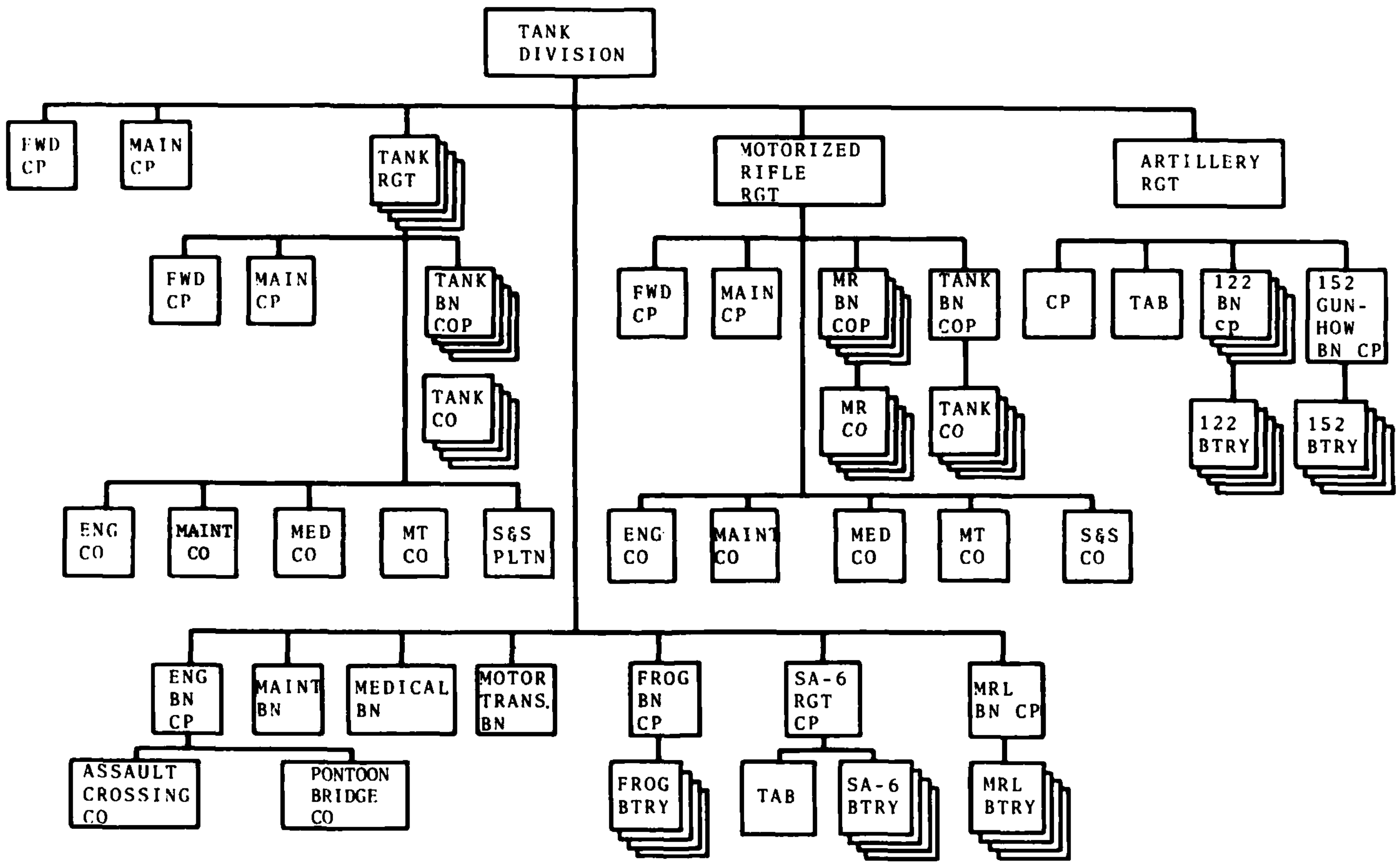


FIGURE II-1
TANK DIVISION

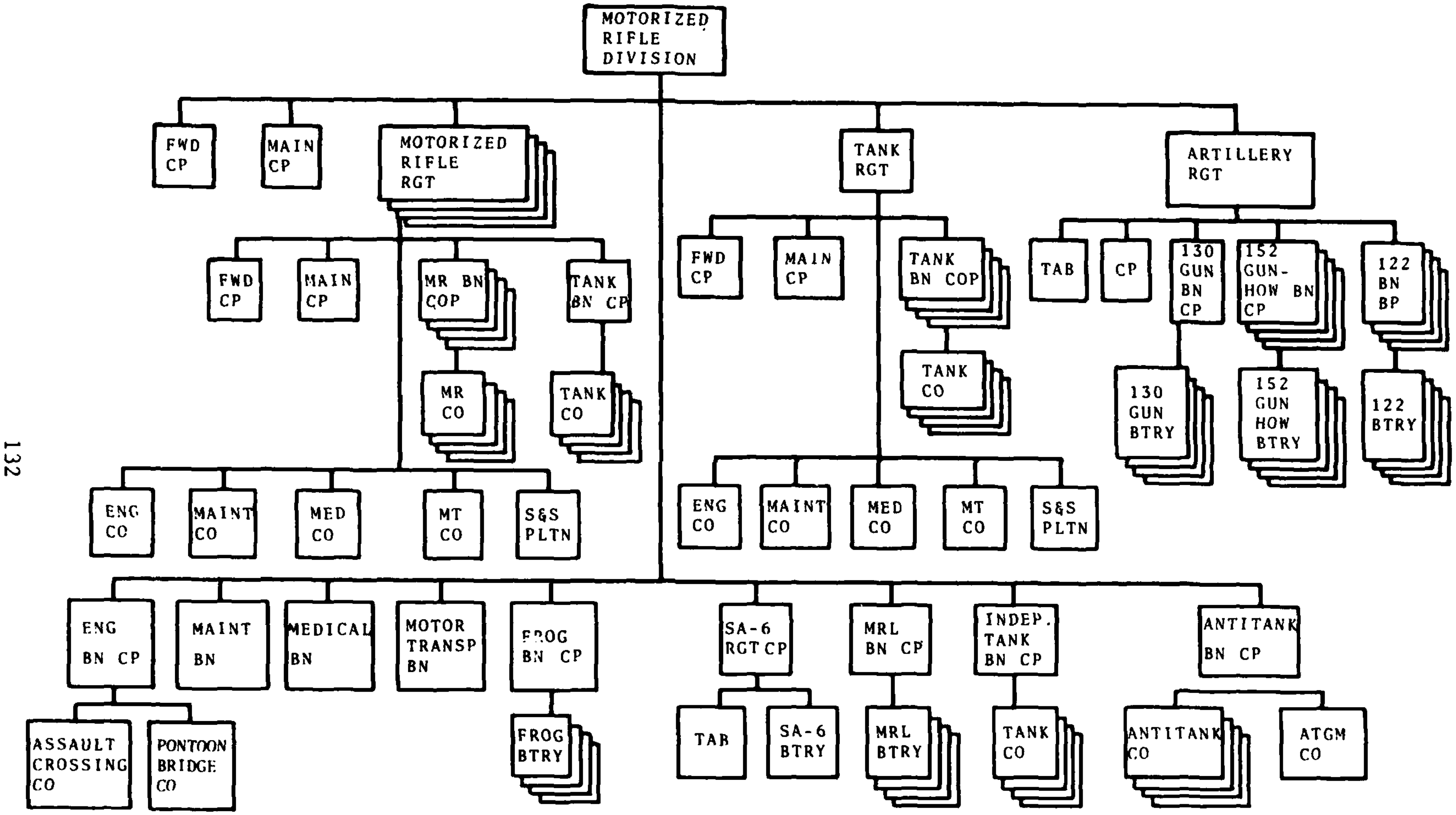


FIGURE II-2
MOTORIZED RIFLE DIVISION

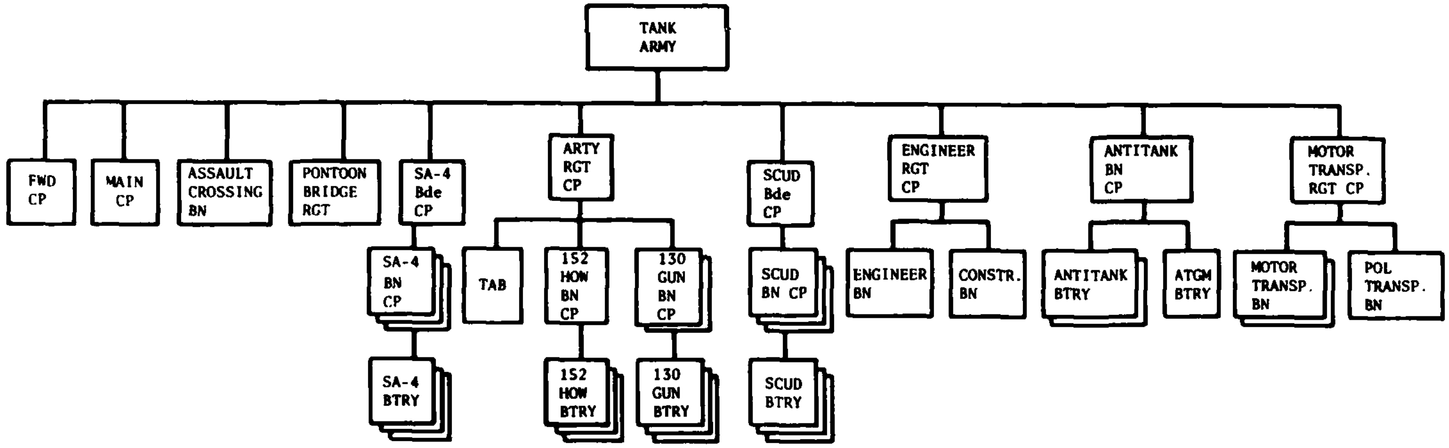


FIGURE II-3
ARMY-LEVEL UNITS

5.0 GENERIC UNIT DATA

This section will describe the data gathered on each generic unit type. (5,7,9,12)

5.1 Equipment

For each unit type, data was gathered on its vehicles, weapons, and radars.

The vehicles were divided into the categories of tread, with four subtypes; artillery, with five subtypes; air defense artillery, with four subtypes, wheeled, with seven subtypes; and engineer, with five subtypes. A file is kept which contains performance characteristics for each of these vehicles. As new units were added to the scenario, any vehicle not listed in the file was included in the category which most closely approximated its characteristics; this was done both to avoid a large file of very similar vehicle characteristics and because performance characteristics were unavailable for some vehicles.

Weapons of a unit type were put into the categories of artillery, tank, or small arms. For purposes of the simulation, the significance of the weapons lies in the rate of ammunition expenditure for the unit and the resulting requirement for resupply, not in the actual weapon count. The ammunition expenditure is expressed as tons per hour per fighting vehicle, and was derived in this way: the weight of an ammunition type's unit of fire (a standard quantity established for measuring amounts of each ammunition type) was multiplied by the standard number of units of fire used per day by a particular type of weapon; this produced the tons per day expenditure,which was divided by 24 to produce the ton-per-hour expenditure for each weapon type; this was expressed as tons-per-hour per fighting vehicle, given the number of weapons of that type on each vehicle.

Radars were categorized as fire control, with two subtypes; surveillance and acquisition, with seven type; or meteorological, with two subtypes.

Although not all these subtypes are represented in the scenario, a file of characteristics was kept on all of them in case they are needed as the simulation is expanded. In some cases, some of the date were unavailable; in those cases, a unit's radars were recorded as belonging to the next most appropriate subtype.

5.2 Radio Nets

A representative radio net structure has been recorded for the two divisions in the scenario. Included are the internal and alternate internal command net, the external command net, and the artillery net, where appropriate for each unit.

5.3 Other Data

Other data for each unit type include the average convoy length by day and night, derived by adding the lengths of the vehicles in a unit and the usual distance between them; the average march speed by day and night when all unit's vehicles are traveling together, derived by determining the average speed of the unit's slowest vehicle; and the average time to clear and close to a new position. This last item was an estimate in many cases, since the data were unavailable for many units.

6.0 FUTURE EFFORTS

Future efforts connected with threat data involve three tasks: 1) the existing data will be incorporated into the Battlefield Environment Module (BEM), which involves conversion from a Pascal-based version of the simulation to a Lisp-based one; 2) the threat representation will continue to evolve through refinements and updates; 3) sources above the SECRET classification should be consulted; and 4) the scope of the data will be extended to the echelons-above-corps level.

REFERENCES

1. The MITRE Corporation, Threat Definition: Unit Locations and Target Arrays for Soviet Air Defense Weapons (U), WP 12588, R. Darron and D. Giles, 1977,(SECRET).

2. The MITRE Corporation, Threat Definition: Unit Locations and Target Arrays for Soviet Combat Service Support Units (U), WP 12843, R. Darron and D. Giles, 1977, (SECRET).

3. The MITRE Corporation, Threat Definition: Unit Locations and Target Arrays for Soviet Artillery Battalions and Maneuver Battalions (U), WP 12724, 1977, R. Darron and D. Giles, , (SECRET).

4. U. S. Army Combined Arms Combat Developments Activity, Standard Scenario for Combat Developments, Europe I, Sequence 2A (U), (SECRET).

5. The MITRE Corporation, C^3 Countermeasures: Initial Analysis of Army Capabilities for the 1980's (U), MTR 80W00278-08, D. Giles and R. Preston, 1983, (SECRET).

6. The MITRE Corporation, Soviet Division Dynamics: Forward Movement of a Second-Echelon Division (U), MTR 07854, R. Darron and D. Giles, 1978, (SECRET).

7. Intelligence and Security Command, and Intelligence and Threat Analysis Center, Soviet Army Operations, 1978.

8. Intelligence and Threat Analysis Center, Functional Area Representation Objectives for the Corps/Division Evaluation Model (CORDIVEM)-RED Side (U), 1982, (SECRET).

9. Defense Intelligence Agency, Soviet Ground Force Organizational Guide (U), 1978, (SECRET).

10. Combined Arms Combat Development Activity, Handbook of Organization and Equipment of the Soviet Army (U), 1980, (SECRET).

11. BDM, Soviet Front and Army Operations (U), 1982, (SECRET).

12. Defense Intelligence Agency, Soviet Ground Force Logistic Guide (U), 1975, (SECRET).

APPENDIX III

DATA LISTS OF THREAT REPRESENTATION FOR THE BBM

(Published Separately)

APPENDIX IV

ROSS-LANGUAGE CODE FOR THE BBM

```
; simulator -- top node

(file-actors: simulator)

(declare (specials t))

(ask something create generic simulator with
   aed                                    'txa4;
   ticksize                               .1     ; simulation ticksize
   ticks-per-ground-update                30
   ticks-per-air-update                   1
   sim-to-real-time-ratio                 5
   window                                 nil
   scenario                               nil
   artist                                 artist
   blue-sensor-control-station-status     nil    ; turned off
   analyst-status                         nil    ; turned off
   current-run-no                         1
   unit-instances                         nil
)

(ask property stop prompting for values)


(ask simulator when receiving (prepare simulation using scenario >scenario)
   (tell simulator initialize using scenario !scenario)
   (tell simulator prepare graphics)
   (tell simulator place actors in initial positions)
)


(ask simulator when receiving (initialize using scenario >scenario)
   (~you set your scenario to !scenario)

   ; set defaults from scenario

   (if (not (equal (ask simulator ask your scenario to recall your ticksize) 'nil)) then
      (~you set your ticksize to !(ask simulator ask your scenario to recall your ticksize)))
   (ask nclock set your $ticksize to !(~your ticksize))

   (if (not (equal (ask simulator ask your scenario to recall your
          ticks-per-ground-update) 'nil)) then
      (~you set your ticks-per-ground-update to
           !(ask simulator ask your scenario to recall your ticks-per-ground-update)))

   (if (not (equal (ask simulator ask your scenario to recall your
               ticks-per-air-update) 'nil)) then
      (~you set your ticks-per-air-update to
           !(ask simulator ask your scenario to recall your ticks-per-air-update)))
```

```
(if (not (equal (ask simulator ask your scenario to recall your
      sim-to-real-time-ratio)  'nil)) then
   (~you set your sim-to-real-time-ratio to
       !(ask simulator ask your scenario to recall your sim-to-real-time-ratio)))

(if (not (equal (ask simulator ask your scenario to recall your window) 'nil)) then
   (~you set your window to !(ask simulator ask your scenario to recall your window)))

(if (not (equal (ask simulator ask your scenario to recall your aed) 'nil)) then
   (~you set your aed to !(ask simulator ask your scenario to recall your aed)))
)
```

```
; draw node network

(ask simulator when receiving (prepare graphics)
   (ask simulator ask your artist to send display to !(~your aed))
   (ask simulator ask your artist to begin painting)
   (ask simulator ask your artist to set window to !(~your window))
   (if (equal (ask !(~your scenario) recall your graphics-scene) 'terrain) then
      (ask simulator ask your artist to draw towns and bridges)
      (ask simulator ask your artist to draw terrain features)
    else
      (ask simulator ask your artist to draw node network)
   )
)


(ask simulator when receiving (place actors in initial positions)


   ; place units in initial scores positions

   (loop for unit-test in (ask unit recall your instances) do
      (if (equal (ask !unit-test recall your object-type) 'instance) then
         (ask simulator add !unit-test to your list of unit-instances)
         (ask simulator ask your artist to place !(ask !unit-test recall your shape) at
                  !(car (ask !unit-test recall your beg-end-loc)))
         (tell !unit-test set your position to
                  !(car (ask !unit-test recall your beg-end-loc)))
         (if (equal 1 (~your current-run-no)) then
            (tell !unit-test set your stored-plan to !(ask !unit-test recall your plan))
         )
      )
   )

   ; place sensor-control-units in initial positions

   (loop for sensor-test in (ask sensor-control-unit recall your instances) do
      (if (equal (ask !sensor-test recall your object-type) 'instance) then
         (ask simulator ask your artist to place !(ask !sensor-test recall your shape) at
                  !(car (ask !sensor-test recall your beg-end-loc)))
         (tell !sensor-test set your position to
                  !(car (ask !sensor-test recall your beg-end-loc)))
         (if (equal 1 (~your current-run-no)) then
            (tell !sensor-test set your stored-plan to !(ask !sensor-test recall your plan))
         )
      )
   )
)


(ask simulator when receiving (start simulation)
   (prog (sim-duration-ticks next-real-tick-time real-ticksize
          sim-print-time blue-station-tick-count)

      (ask nclock set your $stime to 0)

      (ask simulator ask your scenario to prepare to start simulation)

      (setq sim-duration-ticks (add1 (fix (quotient (float (ask !(~your
        scenario) recall your simulation-duration))(float (~your ticksize))))))
```

```
(setq real-ticksize (quotient (float (~your ticksize))
        (float (~your sim-to-real-time-ratio))))

(setq next-real-tick-time (real-time))

(loop as tick-no from 1 to sim-duration-ticks do

   (prog ()
   again
   (if (lessp (real-time) next-real-tick-time) then (go again)))

   ; check for interrupt

   (if interrupt then
      (print-interrupt-message)
      (break)
      (setq next-real-tick-time (real-time))
      (setq interrupt nil)
   )

   (setq sim-print-time (minutes-to-scores
        (fixn 1 (plus ~nclock-time (~your ticksize)))))

   (ask simulator ask your artist to update clock to !sim-print-time)

   ; put out global time for use by analyst and blue sensor control station

   (if (or (~your blue-sensor-control-station-status) (~your  analyst-status)) then
      (puttime sim-print-time))

   (ask nclock tick)


   ;send all ground and air sensor update information (movement ,on
   ;off, etc) (occuring in this tick) to blue sensor control station

   (if (ask sensor-action-unit recall your stored-sensor-updates) then

      (put-out-report (ask sensor-action-unit recall your
         stored-sensor-updates) 'sensrfile 65)

      (ask sensor-action-unit set your stored-sensor-updates to nil)
   )


   ;send all sensor reports (occuring in this tick) to the blue
   ;sensor station

   (if (ask sensor-action-unit recall your stored-sensor-reports) then

      (put-out-report (ask sensor-action-unit recall your
         stored-sensor-reports) 'rptsfile 64)

      (ask sensor-action-unit set your stored-sensor-reports to nil)
   )


   ; send all analyst reports (occuring in this tick) to ANALYST

   (if (ask sensor-action-unit recall your stored-analyst-reports) then
      (put-out-report (ask sensor-action-unit recall your
         stored-analyst-reports) 'anlstfile 67)
      (ask sensor-action-unit set your stored-analyst-reports to nil)
```

```
        )

        ; receive sensor control commands from blue sensor control station

        (query-blue-station)

        (setq next-real-tick-time (plus next-real-tick-time  real-ticksize))
      )
    )
)

; restart simulation from beginning

(ask simulator when receiving (prepare simulation again)
   (loop for unit-test in (ask unit recall your instances) do
     (if (equal (ask !unit-test recall your object-type) 'instance) then
      (ask simulator ask your artist to erase !(ask !unit-test recall your shape) at
              !(ask !unit-test recall your position))
      (ask !unit-test unplan all (+))
      (ask !unit-test set your plan to !(ask !unit-test recall your stored-plan))
     )
   )
   (loop for sensor-test in (ask sensor recall your instances) do
     (if (equal (ask !sensor-test recall your object-type) 'instance) then
       (ask simulator ask your artist to erase !sensor-test sensor coverage)
       (ask simulator ask your artist to erase !(ask !sensor-test recall your shape) at
              !(ask !sensor-test recall your position))
       (ask !sensor-test unplan all (+))
       (ask !sensor-test set your plan to !(ask !sensor-test recall your stored-plan))
       (ask !sensor-test set your cycle-definition to nil)
     )
   )
   (loop for sector-test in (ask sector recall your instances) do
      (ask !sector-test unplan all (+))
      (ask !sector-test set your units to nil)
      (ask !sector-test set your sensors to nil)
   )
   (ask nclock unplan all (+))
   (ask nclock set your $stime to 0)
   (ask simulator place actors in initial positions)
   (ask sensor-action-unit set your sensors-in-use to nil)
   (ask sensor-action-unit set your units-found to nil)
   (ask sensor-action-unit set your stored-sensor-updates to nil)
   (ask sensor-action-unit set your stored-sensor-reports to nil)
   (ask sensor-action-unit set your stored-analyst-reports to nil)
   (ask sector set your units to nil)
   (ask sector set your sensors to nil)
   (ask sector unplan all (+))
   (setq interrupt nil)
   (~you increment your current-run-no by 1)
)


; pass through commands (to artist)

(ask simulator when receiving (display sectors)
   (ask simulator ask your artist to display sectors))

(ask simulator when receiving (erase sectors)
```

```
   (ask simulator ask your artist to erase sectors))


; eliminates all graphics

(ask simulator when receiving (stop graphics)
   (ask simulator set your artist to dead-artist)
)


; restores graphics

(ask simulator when receiving (restart graphics)
   (ask simulator set your artist to artist)
   (ask simulator ask your artist to send display to !(~your aed))
   (ask simulator ask your artist to begin painting)
   (ask simulator ask your artist to set window to !(~your window))
)


(ask simulator when receiving (send reports to blue sensor control station)
   (~you set your blue-sensor-control-station-status to on)
)

(ask simulator when receiving (stop sending reports to blue sensor control station)
   (~you set your blue-sensor-control-station-status to nil)
)

(ask simulator when receiving (send reports to analyst)
   (~you set your analyst-status to on)
)

(ask simulator when receiving (stop sending reports to analyst)
   (~you set your analyst-status to nil)
)
```

```
;; unit -- top node for action and control

(file-actors: unit)

(declare (*fexpr tell*))
(declare (specials t))

(ask simulator create generic unit with
         position      nil   ;(x, y)
         superior      nil   ;superior unit in chain of command
         status        nil   ;can be assembly, contact, river cross,destroyed,
                             ;under attack, moving
         plan          nil   ;sequence of nodes
         beg-end-loc   nil   ;((x1, y1) (x2, y2)) [scores begin. and end pts]
         start-time    nil   ; time when unit leaves initial position
         stop-time     nil   ; time when unit arrives at final position
         vehicles      nil   ;# of vehicles
         speed         nil   ;km/hr
         fuel-cons     nil
         effectiveness nil
         com-delay     .25   ;communications delay
         response-delay .67
         ack-time      .2    ;duration of acknowledgement
         fire-mission-time .5
         fire-mission-call-time   .5
         net-call-dur  .3    ;duration of command call
         clse-call-time .2   ;duration of a closing call
         rpt-time      .2    ;duration of a control point call
         radios        nil
         type          nil   ;generic type
         shape         nil   ;font number for graphics
         cmd-freq      20
         cmd-up-freq   30
         cmd-alt-freq  40
)


(ask unit when receiving (proceed to >position)
  (ask simulator ask your artist to
          move !myself from !(~your position) to !position
          using !(~your shape))
  (~you set your position to !position)
    (if (not (null (~your surveillance))) then (~you turn surveillance radar off))
        (~you unplan all (turn + radar +))
        (ask scheduler unplan all (schedule duty cycle for !myself +))
   (~you set your activity to moving)
  )


(ask unit when receiving (move according to plan)
  (ask simulator ask your artist to
          plan after !(~your com-delay) ~minutes display
          confirmcall comm at !(~your position))
  (ask sector plan after !(~your com-delay) ~minutes !myself has voice
          communication with frequency !(~your cmd-up-freq) and duration
          !(~your ack-time))
  (ask pathfinder move !myself from !(~your position) to
          !(cadr (~your beg-end-loc)))
  )


(ask unit when receiving (have arrived at node >node)
  (ask simulator ask your artist to
```

```
          display controlcall comm at !(ask !node recall your position))
  (tell sector !myself has voice communication with frequency
               !(~your cmd-up-freq) and duration !(~your rpt-time))
  )


(ask unit when receiving (have arrived at final destination)
  (ask simulator ask your artist to display closingcall comm at !(cadr (~your beg-end-loc)))
  (tell sector !myself has voice communication with frequency
               !(~your cmd-up-freq) and duration !(~your close-call-time))
   (~you set your activity to in-position)
   (if (not (null (~your surveillance))) then (ask scheduler-assistant
          schedule duty cycle for !myself surveillance radar))
  )


(ask unit when receiving (turn >type radar >onoff)    ; any unit with radar

  (if (equal (~your activity) 'moving)

         then (set-radar-status myself type 'off)

         (~you unplan all (turn + radar +))

         (ask scheduler unplan all (schedule duty cycle for !myself +))

         else (if (equal onoff 'on)

                        then  (ask simulator ask your artist to display ~the type radar
                                          at !(~your position))
                        (tell sector !myself has turned ~the type radar on
                                   at frequency  !(cadr (memq 'high-freq (getn myself type )))
                                   and duration 15)

                        (set-radar-status myself type 'on)

                        else (ask simulator ask your artist to erase ~the type radar
                                         at !(~your position))
                        (tell sector !myself has turned ~the type radar off)
                        (set-radar-status myself type 'off)
                        )))
```

```
(file-actors: sensor)

(declare (specials t))

(ask simulator create generic sensor with
        position          nil   ;(x, y)
        status            nil   ;active, inactive, jammed, destroyed
        plan              nil
        beg-end-loc       nil   ; ((x1, y1) (x2, y2)) [begin. and end]
        start-time        nil   ; time when sensor leaves initial position
        stop-time         nil   ; time when sensor arrives at final position
        reporting-center  nil
        com-delay         .5
        response-delay    .67
        command-unit      nil
        speed             nil   ;km/hr
        shape             nil   ;font number for graphics
)

(ask sensor when receiving (proceed to >position)
   (ask simulator ask your artist to move !myself from !(~your position) to !position
            using !(~your shape))
   (~you set your position to !position)
)
```

```
(file-actors: control-unit)

(declare (*fexpr tell*))
(declare (specials t))

(ask unit create generic control-unit with
        subordinates    nil   ;subordinate in the chain of command
)


(ask control-unit when receiving (implement battle plan)   ;   all control units

  (prog (time-till-command)

                (ask simulator ask your artist to

                        plan after !(~your com-delay) ~minutes display confirmcall

                        comm at !(~your position))

                (tell sector plan after !(~your com-delay) ~minutes !myself has voice

                         communication with frequency !(~your cmd-up-freq) and duration

                         !(~your ack-time))

                (setq time-till-command (times 2. (~your com-delay)))


                ; pass on orders to military subordinates

                (loop for subordinate in (~your subordinates) do

                         (ask simulator ask your artist to plan after !time-till-command ~minutes display
                                 netcall comm at !(~your position))

                         (tell !subordinate plan after !time-till-command ~minutes
                                      implement battle plan)

                         (tell sector plan after !time-till-command ~minutes !myself has voice
                                      communication with frequency !(~your cmd-freq) and duration
                                      !(~your net-call-dur))
                         )


                ; now move myself

                (if (not (equal (car (~your beg-end-loc)) (cadr (~your beg-end-loc)))) then
                        (~you move according to plan)
                        )
                )
  )


(ask control-unit when receiving (request fire support)   ;   maneuver battalion

  (prog (free-arty-btries num-calls arty-bn arty-btry firing-unit delay-time)

                (tell sector plan after !(setq delay-time
                       (~your response-delay)) ~minutes

                         !myself has voice communication
```

```
          with frequency !(~your cmd-freq)

          and duration !(~your fire-mission-call-time))

      (ask simulator ask your artist to plan after

          !delay-time ~minutes display firecall comm

          at !(~your position))

      (loop for arty-bn in (get (~your superior) 'artillery-support)

          do (loop for arty-btry in (get arty-bn 'subordinates)

                  when   (and

                            (not (equal (get-unit-activity arty-btry) 'moving))
                             (not (equal (get-unit-activity arty-btry) 'firing))

                                               (greaterp
                                                (subl (get-unit-range arty-btry))
                                                (distance-between-2-pts (~your position)
                                                                 (get arty-btry 'position))))
                  do  (setq free-arty-btries
                                               (cons arty-btry free-arty-btries))))

      (if (null free-arty-btries) (return nil))

      (setq firing-unit (pick-unit free-arty-btries))

      (setq num-calls (addl (poisson-dist 2)))

      (setq delay-time (sum delay-time delay-time))
      loop

      (tell !firing-unit plan after !delay-time ~minutes fire mission)

      (ask simulator ask your artist to plan after
               !(add delay-time (~your com-delay)) ~minutes
               display firecall comm at !(~your position))

      (tell sector plan after !(add delay-time (~your com-delay)) ~minutes

          !myself has voice communication

          with frequency !(get firing-unit 'cmd-up-freq)

          and duration !(~your fire-mission-call-time))

      (~let delay-time be (sum delay-time (expon-dist 1.5)))

      (cond ((greaterp (setq num-calls (subl num-calls)) 0)

           (go loop)))))

(ask control-unit when receiving (target acquired)  ; air defense target

  (~block with local variables (firing-unit free-units)

                (loop for firing-unit in (~your subordinates)
```

```
                    and unit-activity = (get-unit-activity firing-unit)
                    when (and (not (equal unit-activity  'firing))
                              (not (equal (getn firing-unit 'type)
                                          'tgt-acq-btry))
                              (not (equal unit-activity 'moving)))
                    do (setq free-units (cons firing-unit free-units)))
        (if (not (null free-units))
            then (tell !(pick-unit free-units) engage the target)
            (~you turn heightfinder radar on)
            (ask sector plan after !(~your com-delay) ~minutes
                 !myself has  voice communication
                 with frequency !(~your cmd-freq)
                 and duration !(~your fire-mission-call-time))
            (ask simulator ask your artist to plan after
                 !(~your com-delay) ~minutes
                 display firecall comm at !(~your position))
            )))

(ask control-unit when receiving (mission complete)  ; sa-6-bn

  (~you plan after !(~your response-delay) ~minutes
             turn heightfinder radar off)

  (ask sector plan after !(~your com-delay) ~minutes
           !myself has voice communication
           with frequency !(~your cmd-freq)
           and duration !(~your ack-time))

  (ask simulator ask your artist to plan after !(~your com-delay)
           ~minutes display firecall comm
           at !(~your position)))
```

```
(file-actors: action-unit)

(declare (*fexpr tell*))
(declare (specials t))


(ask unit create generic action-unit with
          radar     nil
          weapons   nil
          ammo-cons nil
          fire-mission-time .5
          fire-mission-call-time .5
)


(ask action-unit when receiving (implement battle plan)       ;all action units

   (if (not (equal (car (~your beg-end-loc)) (cadr (~your beg-end-loc)))) then
      (~you move according to plan)
   )

)

(ask action-unit when receiving (fire mission)   ;  artillery battery

  (~block with local variables (delay1 delay2)

  (~let delay1 be (sum (~your com-delay) (~your response-delay)))  ;add random

               (~let delay2 be (sum delay1 (~your fire-mission-time)))

  (~you set your activity to firing)


  (~you plan after !delay2
                ~minutes set your activity to in-position)

  (ask sector plan after !(~your com-delay ) ~minutes

               !myself has voice communication

               with frequency !(~your cmd-up-freq)

               and duration !(~your fire-mission-call-time))

  (ask simulator ask your artist to plan after !(~your com-delay) ~minutes

          display firecall comm

          at !(~your position))

  (ask sector plan after !delay2 ~minutes

               !myself has fired artillery)
  ;  question as to whether nclock picks this up in actors running

  (ask simulator ask your artist to plan after

 !delay2 ~minutes display !(get-burst-type myself) burst

          at !(~your position))))
```

```
(ask action-unit when receiving (call for fire)   ;maneuver company in contact
  (ask sector !myself has voice communication
              with frequency !(~your cmd-up-freq)
              and duration !(~your fire-mission-call-time)  )
  (ask simulator ask your artist to display firecall comm
          at !(~your position))
  (ask !(~your superior) request fire support))


(ask action-unit when receiving (fire prep fires)    ; artillery battery
  (~block with local variables (delay)
                (~you set your activity to firing)
                (~let delay be (sum (expon-dist fire-mission-time)
                                             (expon-dist (~your response-delay))))
                (~you plan after !delay ~minutes set your activity to in-position)
                (ask sector plan after !delay ~minutes
                        !myself has fired artillery)
                (ask simulator ask your artist to plan after !delay ~minutes
                        display !(get-burst-type myself) burst
                        at !(~your position))))


(ask action-unit when receiving (engage the target)       ; sa6-btry engagement
  (~block with local variables (delay-time)
                (~let delay-time be (expon-dist (~your com-delay)))
                (ask sector plan after
                        !delay-time  ~minutes
                        !myself has voice communication
                        with frequency !(~your cmd-up-freq)
                        and duration !(~your fire-mission-call-time))
                (ask simulator ask your artist to plan after !delay-time ~minutes
                        display firecall comm
                        at !(~your position))
                (~you set your activity to firing)
                (~let delay-time be
                             (sum delay-time (expon-dist (~your response-delay))))
                (if (greaterp (sum ~nclock-time delay-time) (~your start-time))
```

```
        then (setq delay-time (diff delay-time (~your response-delay)))

        (go loop-out))

(~you plan after (delay-time ~minutes

                turn firecontrol radar on)

(loop for n from 1 to (addl (int-urngen 3))

                do      (~let delay-time be
                               (sum delay-time (expon-dist (~your response-delay))))

                (if (greaterp (sum ~nclock-time delay-time) (~your start-time))

                        then (~let delay-time  be
                                        (diff delay-time (~your response-delay)))

                        (go loop-out))

                (ask sector plan after (delay-time ~minutes
                        (myself has fired missile)

                (ask simulator ask your artist to plan after
                        (delay-time ~minutes
                        display missile burst at
                        ((~your position))))


(~let delay-time be
                (sum delay-time (expon-dist (~your response-delay))))

loop-out                (~you plan after (delay-time

                                        ~minutes turn firecontrol radar off)

(ask ((~your superior) plan after (delay-time ~minutes

        mission complete)

(ask sector plan after (delay-time ~minutes
        (myself has voice communication
        with frequency ((~your cmd-up-freq)
        and duration ((~your ack-time))

(ask simulator ask your artist to plan after
        (delay-time ~minutes display
        firecall comm at
        ((~your position))

(~you plan after (delay-time ~minutes
                set your activity to in-position)))
```

```
(file-actors: sensor-control-unit)

(declare (*fexpr tell*))
(declare (specials t))

(ask sensor create generic sensor-control-unit with
        target-priorities   nil
        shape               43
        ground-sensor-cycle (5 3) ; for active sensors--passive sensors stay on
)


(ask sensor-control-unit when receiving (send ground >type sensor to setup
   point >setup-pt for >mission-duration minute mission)

   (prog (task sensor-type)

      (setq sensor-type (concat type '-g))

      (loop for sensor in (~your subordinates) do
         (if (equal sensor-type (ask !sensor recall your type)) then
            (if (not (member sensor (ask sensor-action-unit recall your
                        sensors-in-use))) then
               (ask !sensor set your beg-end-loc to !(list (car (ask !sensor
                     recall your beg-end-loc)) setup-pt))
               (if (or (equal sensor-type 'mti-g)(equal sensor-type 'cmcb-g)) then
                  (setq task (list '-l mission-duration (car (~your
                       ground-sensor-cycle))(cadr (~your ground-sensor-cycle))))
                else
                  (setq task (list '-l mission-duration mission-duration 0))
               )
               (tell !sensor plan after !(~your com-delay) ~minutes move ground
                     sensor to !setup-pt with !task)
               (go found)
            )
         )
      )
      (print-no-sensor-message)
      found (return)
   )
)


(ask sensor-control-unit when receiving (send up airborne >type sensor in
    >leg-length km racetrack pattern >num-circuits times around with
    initial pt >init-pt)

   (prog (task sensor-type plan mission-duration leg-time)

      (setq sensor-type (concat type '-a))

      (loop for sensor in (~your subordinates) do
         (if (equal sensor-type (ask !sensor recall your type)) then
            (if (not (member sensor (ask sensor-action-unit recall your
                          sensors-in-use))) then

               (setq leg-time (times 60. (quotient (float leg-length)
                                 (float (ask !sensor recall your speed)))))

               (setq mission-duration (times num-circuits
                                        (plus 2. (times 2. leg-time))))
```

```
            (ask !sensor set your num-circuits to !num-circuits)

            (setq task (list '-1 mission-duration leg-time

               1))  ; use this value to turn sensor on for both legs

             ; (plus 2. leg-time))) ; use this value for only right leg

            (setq plan (list leg-length num-circuits init-pt))

            (tell !sensor plan after !(~your com-delay) ~minutes move
               airborne sensor in racetrack pattern with !task and !plan)
            (go found)
         )
      )
   )
   (print-no-sensor-message)
   found (return)
 )
)

(ask sensor-control-unit when receiving (send up airborne imint sensor to points +points)
  (prog ()
    (loop for sensor in (~your subordinates) do
      (if (equal (ask !sensor recall your type) 'imint-a) then
        (if (not (member sensor (ask sensor-action-unit recall your sensors-in-use))) then
          (tell !sensor plan after !(~your com-delay) ~minutes move airborne
                imint sensor along points &points)
          (go found)
        )
      )
    )
    (print-no-sensor-message)
    found (return)
  )
)
```

```
(file-actors: sensor-action-unit)

(declare (*fexpr tell*))
(declare (specials t))

(ask sensor create generic sensor-action-unit with
       cycle-definition  nil    ;(mission-start-time, mission-duration,
                                ; cycle-on-duration, cycle-off-duration)
       current-status    nil
       range             nil
       frequency         nil
       saturation-level  nil
       vulnerability     nil    ;related to vehicle speed
       listening-mode    nil    ;ground/airborne listening
       type              nil
       units-found       nil    ;list of unit id's found by this sensor
       sector-sensors    (mti-g mti-a elint-g elint-a) ;sensors requiring sectorization
       nonsector-sensors (comint-a comint-g cmcb-g)  ; sensors not requiring sectorization
       sensors-in-use    nil    ;list of sensor id's for all sensors which
                                ;have started their missions
       stored-sensor-updates nil ;sent out to blue sensor control station
       stored-sensor-reports nil ;sent out to blue sensor control station
       stored-analyst-reports nil ;send out to ANALYST
       sectorization-switch-range 10 ; less than this, use sectors--- greater,
                                     ; use global sector

)
(ask sensor-action-unit when receiving(move ground sensor to >point with >task)

   (~you set your cycle-definition to |task)

   (~you set your position to |(car (~your beg-end-loc)))
   (ask simulator ask your artist to place |(~your shape) at |(~your position)) ;sensor has received
                              ; first message and is now relevant for display

   (if (not (equal (car (~your beg-end-loc)) (cadr (~your beg-end-loc)))) then
      (ask pathfinder move ground sensor |myself from |(~your position) to |point)
    else
      (~you set your stop-time to |~nclock-time)
   )
   (ask mathematician reconstruct cycle definition for |myself)


   (ask sensor-action-unit add |myself to your list of sensors-in-use)
   (ask sensor-action-unit plan after |(diff (cadar (last
      (~your cycle-definition))) ~nclock-time) ~minutes remove |myself from
      your list of sensors-in-use)


   ; determine sector penetration of sensor's coverage (if relevant)

   (if (member (~your type) (ask sensor-action-unit recall your sector-sensors)) then
      (if (equal (~your type) 'mti-g) then
         (ask mathematician determine sector penetration of ground |myself)
       else  ; sensor is elint-g -- use global sector
         (ask sector plan after |(diff (car (~your cycle-definition)) ~nclock-time)
            ~minutes |myself has penetrated the sector with |(list myself
            'elint-g (caar (~your cycle-definition)) (cadar (~your cycle-definition))))
      )
   )

   ; send messages to artist to turn sensor on and off according to its
```

```
; task (duty cycle) and messages(indirectly) to blue sensor control station

(loop for on-interval in (~your cycle-definition) do

  (ask simulator ask your artist to plan after !(diff (car on-interval)
     ~nclock-time) ~minutes display !myself sensor coverage)

  (ask simulator ask your artist to plan after !(diff (cadr on-interval)
     ~nclock-time) ~minutes erase !myself sensor coverage)

  (if (ask simulator recall your blue-sensor-control-station-status) then
     (ask sensor-action-unit plan after !(diff (car on-interval)
         ~nclock-time) ~minutes add !(list myself
         (cadr (~your beg-end-loc)) 'on) to your list of stored-sensor-updates)


     (ask sensor-action-unit plan after !(diff (cadr on-interval)
        ~nclock-time) ~minutes add !(list myself
        (cadr (~your beg-end-loc)) 'off) to your list of stored-sensor-updates)
  )
 )
)


(ask sensor-action-unit when receiving (move airborne sensor in racetrack
       pattern with >task and >plan)
   (~you set your current-status to off)
   (~you set your cycle-definition to !task)


   (~you set your plan to !plan)
   (~you set your position to !(car (~your beg-end-loc)))
   (ask simulator ask your artist to place !(~your shape) at !(~your position)) ; sensor has
                  ;   received first message and is now relevant for display

   (ask flightplanner move airborne sensor !myself using racetrack pattern)
   (ask mathematician reconstruct cycle definition for !myself)


   (if (member (~your type)(ask sensor-action-unit recall your sector-sensors)) then
      (if (and (lessp (~your range) (~your sectorization-switch-range))
               (equal (~your type) 'mti-a)) then
         (ask mathematician determine sector penetration of airborne sensor
              !myself with task !task)
       else  ; sensor is elint or sensor is mti and
              ; range is large enough to cover most sectors--use only global sector
          (ask mathematician determine penetration of global generic sector with
              airborne sensor !myself)
      )
   )

   (ask sensor-action-unit add !myself to your list of sensors-in-use)
   (ask sensor-action-unit plan after !(diff (~your stop-time)
       ~nclock-time) ~minutes remove !myself from your list of sensors-in-use)

)



(ask sensor-action-unit when receiving (fly to >position)
   (prog (current-time found former-position)
      (setq former-position (~your position))
```

```
(~you set your position to |position)

(setq found nil)
(setq current-time ~nclock-time)
(loop for duty-cycle in (~your cycle-definition) do
   (if (and (greaterp current-time (car duty-cycle))
            (lessp current-time (cadr duty-cycle))) then
      (setq found t)
      (return t)
   )
)

(if found then
   (if (equal (~your current-status) 'off) then
      (~you set your current-status to on)
   )
   (ask simulator ask your artist to erase |myself sensor coverage)
   (ask simulator ask your artist to move |myself from |former-position
      to |position using |(~your shape))
   (ask simulator ask your artist to display |myself sensor coverage)

 else
   (if (equal (~your current-status) 'on) then
      (~you set your current-status to off)
      (ask simulator ask your artist to erase |myself sensor coverage)
   )
   (ask simulator ask your artist to move |myself from |former-position to |position
            using |(~your shape))
)

; save report for blue sensor control station

(if (ask simulator recall your blue-sensor-control-station-status) then
   (ask sensor-action-unit add |(list myself position
      (~your current-status)) to your list of stored-sensor-updates)
)
  )
 )
```

```
; Moving Target Indicator Sensor

(file-actors: mti)

(declare (specials t))

(ask sensor-action-unit create generic mti with
   time-to-clear 0
   time-to-close 0
   day-speed  35
   night-speed 30
)


(ask mti when receiving (check for sensor detection of >unit)

   ; this will be the formal sensor model for moving targets. At the moment it
   ; is not implemented. Since it is entered after a time and space overlap
   ; has been determined by sector/mathematician it will for the moment report
   ; out 100% of those overlaps as detections.

   (tell interface mti !myself has detected !unit)

)
```

```
; Communications Intelligence Sensor

(file-actors: comint)

(declare (specials t))

(ask sensor-action-unit create generic comint with
   coverage-theta  360
   time-to-clear 0
   time-to-close 0
   night-speed  30
   day-speed    35
)


(ask comint when receiving (check for sensor detection of >unit at frequency
         >freq and duration >duration)

   ; this will be the formal sensor mode for comint.  At the moment it is not
   ; implemented.  Since it is entered after a time and space overlap has been
   ; determined by sector/mathematician it will for the moment report out
   ; 100% of these overlaps as detections.

   (if (and (greater-eq-p freq (car (~your freq-range)))
                      (less-eq-p freq (cadr (~your freq-range)))) then
      (tell interface comint !myself has detected !unit at frequency !freq
           and duration !duration)
   )

)
```

```
; Counter Mortar / Counter Battery Sensor

(file-actors: cmcb)

(declare (specials t))

(ask sensor-action-unit create generic cmcb with
   coverage-theta 45
   time-to-clear 0
   time-to-close 0
   night-speed 30
   day-speed 35
)

(ask cmcb when receiving (check for sensor detection of >unit)

   ; this will be the formal sensor model for cmcb.  At the moment it is not
   ; implemented.  Since it is entered after a time and space overlap has been
   ; determined by sector/mathematician it will for the moment report out
   ; 100% of these overlaps as detections.

   (tell interface cmcb !myself has detected !unit)

)
```

```
; elint sensor

(file-actors: elint)

(declare (specials t))

(ask sensor-action-unit create generic elint with
   coverage-theta  45
   time-to-clear 0
   time-to-close 0
   night-speed 30
   day-speed 35
)

(ask elint when receiving (check for sensor detection of )unit of type )type
   and frequency )freq)

   ; this will be the formal sensor model for elint.  At the moment it is not
   ; implemented.  Since it is entered after a time and space overlap has been
   ; determined by sector/mathematician it will for the moment report out
   ; 100% of these overlaps as detections.

   (tell interface elint !myself has detected !unit of type !type and
       frequency !freq)

)
```

```
; imint sensor behaviors

(file-actors: imint)

(declare (*fexpr tell*))
(declare (specials t))

(ask sensor-action-unit create generic imint with
   reporting-delay  5
)

(ask imint when receiving (check for sensor detection of >unit)

   ; future formal sensor model

   (tell interface plan after !(~your reporting-delay) ~minutes
              imint !myself has detected !unit at time !~nclock-time)

)


(ask imint when receiving (move airborne imint sensor along points >points)
   (~you set your current-status to off)
   (~you set your position to !(car (~your beg-end-loc)))

   (ask simulator ask your artist to place !(~your shape) at !(~your position))
   (ask flightplanner move airborne imint sensor !myself along points !points)
   (ask sensor-action-unit add !myself to your list of sensors-in-use)
   (ask sensor-action-unit plan after !(diff (~your stop-time) ~nclock-time)
      ~minutes remove !myself from your list of sensors-in-use)
)

(ask imint when receiving (fly to >position)
   (prog (former-position)
      (setq former-position (~your position))
      (~you set your position to !position)
      (if (equal (~your current-status) 'on) then
         (ask simulator ask your artist to move !myself from !former-position to
                !position using !(~your shape))
         (ask simulator ask your artist to display !myself sensor coverage)
         (if (ask simulator recall your blue-sensor-control-station-status) then
            (ask sensor-action-unit add !(list myself position
               (~your current-status)) to your list of stored-sensor-updates)
         )
         (~you set your current-status to off)
       else  ; current-status is off
         (ask simulator ask your artist to erase !myself sensor coverage)
         (ask simulator ask your artist to move !myself from !former-position to
            !position using !(~your shape))
         (if (ask simulator recall your blue-sensor-control-station-status) then
            (ask sensor-action-unit add !(list myself position
                (~your current-status)) to your list of stored-sensor-updates)
         )
      )
   )
)
```

```
(file-actors: mathematician)

(declare (*fexpr tell*))
(declare (specials t))

(ask simulator create instance mathematician)


(ask mathematician when receiving (determine time/space overlaps between
    >red-radar on for >duration minutes and any elint sensors)
   (prog (time-overlap-start time-overlap-end time-space-overlaps
            detection-time red-radar-end-time)
      (setq time-overlap-start ~nclock-time)
      (setq red-radar-end-time (plus time-overlap-start duration))
      (loop for elint-test in (ask sector recall your elint-sensors) do
         (setq time-overlap-end (min red-radar-end-time (cadddr elint-test)))
         (setq detection-time (elint-radar-overlap (car elint-test)
            red-radar time-overlap-start time-overlap-end))
         (if detection-time then
            (setq time-space-overlaps (append time-space-overlaps (list (list
               (car elint-test) detection-time))))
         )
      )
      (return time-space-overlaps)
   )
)


(ask mathematician when receiving (determine time/space overlaps between
   elint >sensor-data and any red radars)
   (prog (time-overlap-start time-overlap-end time-space-overlaps detection-time sensor-end-time)
      (setq time-overlap-start (caddr sensor-data))
      (setq sensor-end-time (cadddr sensor-data))
      (loop for radar-test in (ask sector recall your active-red-radars) do
         (setq time-overlap-end (min sensor-end-time (caddddr radar-test)))
         (setq detection-time (elint-radar-overlap (car sensor-data)
            (car radar-test) time-overlap-start time-overlap-end))
         (if detection-time then
            (setq time-space-overlaps (append time-space-overlaps (list (list
               (car radar-test) detection-time (cadr radar-test) (caddr radar-test)))))
         )
      )
      (return time-space-overlaps)
   )
)


; this section plans the moves of the objects for graphics updates

(ask mathematician when receiving (calculate update points for >object)

   (prog  (node update-interval update-time cur-sim-time stop-time object-plan
            update-position current-pos next-pos beg-end-loc transfer
            ticks-per-update)

      (setq beg-end-loc (ask !object recall your beg-end-loc))

      (if (equal 'a (car (reverse (explode (ask !object recall your type)))))
        then
         (setq transfer 'fly)
         (setq ticks-per-update
                  (ask simulator recall your ticks-per-air-update))
```

```
      else
       (setq transfer 'proceed)
       (setq ticks-per-update
                 (ask simulator recall your ticks-per-ground-update))
     )


     (setq update-interval (times ticks-per-update (ask nclock recall your
              $ticksize)))
     (setq object-plan (ask |object recall your plan)) ;now of format(t,(x,y))
     (setq update-time (ask |object recall your start-time))
     (setq cur-sim-time ~nclock-time)
     (setq stop-time (ask |object recall your stop-time))
     (setq current-pos (list update-time (car beg-end-loc)))
     (setq next-pos (car object-plan))

     again

     (setq update-time (plus update-time update-interval))
     (if (greaterp update-time stop-time) then
        (setq update-position (cadr beg-end-loc))
        (setq update-time stop-time)
        (ask |object plan after |(diff update-time cur-sim-time) ~minutes
           |transfer to |update-position)
        (return t)
      else (if (not (greaterp update-time (car next-pos))) then
              (setq update-position (ratio-pt-xy update-time current-pos
                                          next-pos))
              (ask |object plan after |(diff update-time cur-sim-time) ~minutes
                  |transfer to |update-position)
              (go again)
           else (if (greaterp update-time (caar (last object-plan))) then
              (setq current-pos (car (last object-plan)))
              (setq next-pos (list stop-time (cadr beg-end-loc)))
              (setq update-position (ratio-pt-xy update-time current-pos
                            next-pos))
              (ask |object plan after |(diff update-time cur-sim-time)
                              ~minutes |transfer to |update-position)
              (go again)
           else  ; update-time > (car next-pos)
              (setq current-pos next-pos)
              (setq object-plan (cdr object-plan))
              (setq next-pos (car object-plan))
              (setq update-position (ratio-pt-xy update-time current-pos
                      next-pos))
              (ask |object plan after |(diff update-time cur-sim-time) ~minutes
                     |transfer to |update-position)
              (go again)
                 )
           )
      )
    )
 )


(ask mathematician when receiving (determine time and space overlaps between
  >unit-data and any sensors in >sector)

   (prog (unit-exit-time time-overlap-end time-overlap-start
         time-space-overlaps unit-id detection-time )
      (setq unit-id (car unit-data))
      (setq time-overlap-start (caddr unit-data))
      (setq unit-exit-time (caadddr unit-data))
```

```
      (loop for sensor-test in (ask !sector recall your sensors)
       do  ; every sensor on list has a time overlap with this unit
         (if (not (member unit-id (ask !(car sensor-test)
            recall your units-found))) then ; unit has not already been found by
                                            ; this sensor in a previous sector
            (setq time-overlap-end (min unit-exit-time (cadddr sensor-test)))

            ; check for space overlap by determining if any nodes between
            ; time-overlap-start & time-overlap-end are within sensor's coverage

            (setq detection-time (any-pt-in-overlap-in-fan unit-data
                                 sensor-test time-overlap-start time-overlap-end))
            (if detection-time then
               (setq time-space-overlaps (append time-space-overlaps (list (list
                                            (car sensor-test) detection-time))))
               (tell !(car sensor-test) add !(car unit-data) to your list of
                   units-found)
            )
         )
      )
      (return time-space-overlaps)
   )
)




(ask mathematician when receiving (determine time and space overlaps between
  >sensor-data and any units in >sector)

   (prog (time-overlap-end time-overlap-start sensor-end-time
          time-space-overlaps detection-time)
      (setq time-overlap-start (caddr sensor-data))
      (setq sensor-end-time (cadddr sensor-data))
      (loop for unit-test in (get sector 'units) do
         ; every unit on list has a time overlap with this sensor
         (setq time-overlap-end (min (caadddr unit-test) sensor-end-time))

         ; check for space overlap by determining if any nodes between
         ; time-overlap-start & time-overlap-end are within sensor's coverage

         (setq detection-time (any-pt-in-overlap-in-fan unit-test sensor-data
                                 time-overlap-start time-overlap-end))
         (if detection-time then
            (setq time-space-overlaps (append time-space-overlaps (list (list
                                 (car unit-test) detection-time))))
            (tell !(car sensor-data) add !(car unit-test) to your list of
                                 units-found)
         )
      )
      (return time-space-overlaps)
   )
)


(ask mathematician when receiving (reconstruct cycle definition for >object)
   (prog (start-time stop-time mission-start-time mission-end-time
          cycle-on-duration cycle-off-duration cycle-def)

      ;initialize
```

```
(setq cycle-def (ask !object recall your cycle-definition))

(if (equal (car cycle-def) -1) then ; indicates calculate the start time
   (if (equal (car (reverse (explode (ask !object recall your type))))
                        'g ) then  ; ground sensor
      (setq mission-start-time (plus .0001
                          (ask !object recall your stop-time)))
    else  ; airborne sensor
      (setq mission-start-time (caar (ask !object recall your plan)))
   )
 else
   (setq mission-start-time (car cycle-def))
)

(setq mission-end-time (plus mission-start-time (cadr cycle-def)))
(setq cycle-on-duration (caddr cycle-def))
(setq cycle-off-duration (cadddr cycle-def))
(setq stop-time (plus mission-start-time cycle-on-duration))
(setq cycle-def (list (list mission-start-time stop-time)))

again

(setq start-time (plus stop-time cycle-off-duration))
(setq stop-time (plus start-time cycle-on-duration))

(if (greaterp stop-time mission-end-time) then
   (ask !object set your cycle-definition to (cycle-def)
   (return t)
)
(setq cycle-def (append cycle-def (list (list start-time stop-time))))
(go again)
  )
)

(ask mathematician when receiving (determine time/space overlaps between >unit
                   and any comint sensors)

   (prog (current-time sensor-pos time-space-overlaps on-interval)

      (setq current-time ~nclock-time)
      (loop for sensor-test in (ask sensor-action-unit recall your
                                    sensors-in-use) do
         (if (equal 'comint (implode (reverse (cddr (reverse (explode
               (ask !sensor-test recall your type))))))) then ;is comint sensor
            (loop for on-interval in (ask !sensor-test recall your
                    cycle-definition) do
               (if (and (greater-eq-p current-time (car on-interval))
                        (less-eq-p current-time (cadr on-interval))) then
                        ; sensor is turned on at current time

                  ; check for time and space overlap

                  (setq sensor-pos (position-at-time-t sensor-test
                                         current-time))

                  (if (point-within-circle (position-at-time-t unit
                          current-time) (car sensor-pos) (cadr sensor-pos)
                          (square (ask !sensor-test recall your range))) then
                      (setq time-space-overlaps (append time-space-overlaps
                                   (list sensor-test)))
```

```
                  )
               )
            )
         )
         terminate
         (return time-space-overlaps)  ; list of sensor id's
      )
   )


(ask mathematician when receiving (determine time/space overlaps between >unit
                         and any cmcb sensors)

   (prog (current-time sensor-pos time-space-overlaps on-interval)

      (setq current-time ~nclock-time)
      (loop for sensor-test in (ask sensor-action-unit recall your
                                       sensors-in-use) do
         (if (equal 'cmcb-g (ask !sensor-test recall your type)) then
            (loop for on-interval in (ask !sensor-test recall your
                    cycle-definition) do
               (if (and (greater-eq-p current-time (car on-interval))
                        (less-eq-p current-time (cadr on-interval))) then
                        ; sensor is turned on at current time

                  ; check for time and space overlap

                  (setq sensor-pos (position-at-time-t sensor-test
                                      current-time))

                  (if (point-within-fan (position-at-time-t unit
                          current-time) (car sensor-pos) (cadr sensor-pos)
                          (square (ask !sensor-test recall your range))
                          (times .5 (ask !sensor-test recall your coverage-theta))) then
                      (setq time-space-overlaps (append time-space-overlaps
                                  (list sensor-test)))
                  )
                  (go terminate)
               )
            )
         )
      )
      terminate
      (return time-space-overlaps)  ; list of sensor id's
   )
)



;****************************************************************************
;*                                                                          *
;* All behaviors beyond this point concern sector penetration calculations *
;*                                                                          *
;****************************************************************************


(ask mathematician when receiving (determine sector penetration of airborne
   sensor >sensor with task >task)

   (prog (turn-on-time-up turn-off-time-up fan-max-y fan-min-y circuit-time
          max-fan-width fan-min-x-up fan-max-x-up fan-min-x-down fan-max-x-down
```

```
        hp-min-y hp-max-y fan-half-width sector-min-y sector-max-y
        sector-min-x sector-max-x principal-pt-x sensor-range header
        half-theta square-range sensor-data fan-min-y-init-up
        turn-on-time-down turn-off-time-down fan-min-y-init-down
        fan-max-y-init-down fan-max-y-init-up)

    (setq sensor-range (ask |sensor recall your range))
    (setq fan-min-x-up (caadar (ask |sensor recall your plan)))
    (setq fan-max-x-up (plus fan-min-x-up sensor-range))
    (setq fan-min-x-down (caadadddr (ask |sensor recall your plan)))
    (setq fan-max-x-down (plus fan-min-x-down sensor-range))
    (setq hp-min-y (cadadar (ask |sensor recall your plan)))
    (setq hp-max-y (cadadadr (ask |sensor recall your plan)))
    (setq half-theta (times .5 (ask |sensor recall your coverage-theta)))
    (setq square-range (square sensor-range))
    (setq max-fan-width (times sensor-range (sind half-theta)))
    (setq fan-min-y (diff hp-min-y max-fan-width))
    (setq fan-max-y (plus hp-max-y max-fan-width))
    (setq circuit-time (quotient (float (cadr task))(float (ask |sensor recall your num-circuits))))
    (setq header (list sensor (ask |sensor recall your type)))
    (setq turn-on-time-down (caadddr (ask |sensor recall your plan)))
    (setq turn-off-time-down (caaddddr (ask |sensor recall your plan)))
    (setq fan-min-y-init-down (diff hp-max-y max-fan-width))
    (setq fan-max-y-init-down (plus hp-max-y max-fan-width))
    (setq turn-on-time-up (caar (ask |sensor recall your plan)))
    (setq turn-off-time-up (caadr (ask |sensor recall your plan)))
    (setq fan-min-y-init-up (diff hp-min-y max-fan-width))
    (setq fan-max-y-init-up (plus hp-min-y max-fan-width))

    (loop for sector-test in (ask sector recall your descendants) do

       (setq sector-min-x (caar (ask |sector-test recall your position)))
       (setq sector-max-x (caar (last (ask |sector-test recall your
             position))))
       (setq sector-min-y (cadar (ask |sector-test recall your position)))
       (setq sector-max-y (cadadr (ask |sector-test recall your position)))

       ; going up right side of holding pattern

       (if (not (or (greater-eq-p fan-min-x-up sector-max-x)
                 (less-eq-p fan-max-x-up sector-min-x)
                 (less-eq-p fan-max-y sector-min-y)
                 (greater-eq-p fan-min-y sector-max-y))) then

          (setq sensor-data (record-penetration
               'up sector-test fan-min-x-up hp-min-y square-range
               half-theta header turn-on-time-up turn-off-time-up
               sensor fan-max-y-init-up fan-max-x-up fan-min-y-init-up
               sector-min-y sector-max-y hp-min-y hp-max-y
               sensor-range sector-max-x sector-min-x max-fan-width))

          ; send penetration messages for first and succeeding revolutions

          (loop as n from 1 to (getn sensor 'num-circuits) do
             (tell |sector-test plan after |(diff (caddr sensor-data)
                ~nclock-time) ~minutes |sensor has penetrated
                the sector with |sensor-data)

             (setq sensor-data (append header (list (plus (caddr sensor-data)
                  circuit-time)(plus (cadddr sensor-data) circuit-time))))
          )
       )
```

```
      ; going down left side of holding pattern

      (if (not (or (greater-eq-p fan-min-x-down sector-max-x)
                   (less-eq-p fan-max-x-down sector-min-x)
                   (less-eq-p fan-max-y sector-min-y)
                   (greater-eq-p fan-min-y sector-max-y))) then

         (setq sensor-data (record-penetration
            'down sector-test fan-min-x-down hp-max-y square-range
            half-theta header turn-on-time-down turn-off-time-down
            sensor fan-max-y-init-down fan-max-x-down fan-min-y-init-down
            sector-min-y sector-max-y hp-min-y hp-max-y
            sensor-range sector-max-x sector-min-x max-fan-width))

         ; send penetration messages for first and succeeding revolutions

         (loop as n from 1 to (getn sensor 'num-circuits) do
            (tell !sector-test plan after !(diff (caddr sensor-data)
               ~nclock-time) ~minutes !sensor has penetrated
               the sector with !sensor-data)

            (setq sensor-data (append header (list (plus (caddr sensor-data)
                 circuit-time)(plus (cadddr sensor-data) circuit-time))))
         )
       )
     )
   )
 )


(ask mathematician when receiving (determine sector penetration for >unit)

   (prog (unit-data header current-sector current-node transition-point
              next-sector unit-plan begin-pt end-pt)

      (setq unit-plan (ask !unit recall your plan))

      ; form augmented unit plan with the begin and end locations attached
      ; to the fron and back of the unit plan
      ; for the moment consider the sector that the begin pt is in is the same
      ; as the sector for the first node and that the sector that the end pt
      ; is in is the same as the sector for the last node

      (setq begin-pt (list (ask !unit recall your start-time)
                           (cadar unit-plan)))
      (setq end-pt (list (ask !unit recall your stop-time)
                           (cadar (last unit-plan))))
      (setq unit-plan (append (append (list begin-pt) unit-plan)(list end-pt)))


      (setq header (list unit (car (ask !unit recall your parents))))

      ; determine information for first sector

      (setq current-node (car unit-plan))
      (setq current-sector (ask !(cadr current-node) recall your sector))

      ;following unit-data will be sent to sector when first exit is determined

      (setq unit-data (append header (list (cons (car current-node)
         (ask !(cadr current-node) recall your position)))))
```

```
   ; now loop through the nodes of the unit's plan

   (loop for next-node in (cdr unit-plan)
      do (setq next-sector (ask !(cadr next-node) recall your sector))
         (if (not (equal next-sector current-sector)) then ;crossed boundary
               (setq transition-point (time-distance-midpoint
               (cons (car current-node)
               (ask !(cadr current-node) recall your position))
               (cons (car next-node) (ask !(cadr next-node) recall your
               position))))

               (setq unit-data (append unit-data (list transition-point)))
               (tell !current-sector plan after !(difference (caaddr
                 unit-data) ~nclock-time) ~minutes !unit
                 has penetrated sector with !unit-data)
               (setq unit-data (append header (list transition-point)))
         )

      ; prepare for next loop

      (setq current-node next-node)  ; node time and name
      (setq current-sector next-sector)
   )

   ; special for last sector

   (setq unit-data (append unit-data (list (cons (car current-node)
      (ask !(cadr current-node) recall your position)))))
   (tell !current-sector plan after !(diff (caaddr unit-data) ~nclock-time)
      ~minutes !unit has penetrated sector with !unit-data)

   ; send information to the global (generic) sector

   (tell sector plan after !(diff (ask !unit recall your start-time)
      ~nclock-time) ~minutes !unit has penetrated sector with !(append header
      (list (list (ask !unit recall your start-time)(list nil nil))
            (list (ask !unit recall your stop-time)(list nil nil)))))

  )
)


(ask mathematician when receiving (determine sector penetration of
                                    ground >sensor)
  (prog (sensor-position xcoord-sensor-pos ycoord-sensor-pos
        fan-max-x fan-min-x fan-max-y fan-min-y sensor-range header half-theta
            square-range sensor-data max-fan-half-width principal-pt-x)

     (setq sensor-position (cadr (ask !sensor recall your beg-end-loc)))
     (setq xcoord-sensor-pos (car sensor-position))
     (setq ycoord-sensor-pos (cadr sensor-position))
     (setq sensor-range (ask !sensor recall your range))
     (setq square-range (square sensor-range))
     (setq half-theta (times .5 (ask !sensor recall your coverage-theta)))
     (setq fan-min-x xcoord-sensor-pos)
     (setq fan-max-x (plus fan-min-x sensor-range))
     (setq max-fan-half-width (times sensor-range (sind half-theta)))
     (setq fan-min-y (diff ycoord-sensor-pos max-fan-half-width))
     (setq fan-max-y (plus ycoord-sensor-pos max-fan-half-width))
     (setq header (list sensor (ask !sensor recall your type)))
     (setq principal-pt-x (plus fan-min-x (times sensor-range (cosd half-theta))))
```

```
(loop for sector-test in (ask sector recall your descendants) do
   (prog (sector-min-x sector-max-x sector-min-y sector-max-y found)
      (setq sector-min-x (caar (ask !sector-test recall your position)))
      (setq sector-max-x (caar (last (ask !sector-test recall your position))))
      (setq sector-min-y (cadar (ask !sector-test recall your position)))
      (setq sector-max-y (cadadr (ask !sector-test recall your position)))
      (setq sensor-data nil)
      (setq found nil)

      ; run through filter first

      (if (not (or (greater-eq-p fan-min-x sector-max-x)
                   (less-eq-p fan-max-x sector-min-x)
                   (less-eq-p fan-max-y sector-min-y)
                   (greater-eq-p fan-min-y sector-max-y))) then

         ; determine if sector-coordinates are within fan

         (loop for coord in (ask !sector-test recall your position) do
            (if (point-within-fan coord xcoord-sensor-pos
                  ycoord-sensor-pos square-range half-theta) then
               (setq found t)
               (return)
            )
         )

         ; determine if fan extreme pts are within sector

         (if (not found) then
            (if (points-within-sector (list sensor-position
                   (list principal-pt-x fan-max-y)
                   (list fan-max-x ycoord-sensor-pos)
                   (list principal-pt-x fan-min-y)) sector-test) then
               (setq found t)
            )
         )

         (if found then
            (loop for on-interval in (ask !sensor recall your cycle-definition) do
               (setq sensor-data (append header (list (car on-interval)
                                         (cadr on-interval))))
               (tell !sector-test plan after !(diff (car on-interval) ~nclock-time)
                  ~minutes !sensor has penetrated the sector with !sensor-data)
            )
         )
      )
    )
  )
 )
)

(ask mathematician when receiving (determine penetration of global generic
   sector with airborne sensor >sensor)

   (loop for on-interval in (ask !sensor recall your cycle-definition) do
      (tell sector plan after !(diff (car on-interval) ~nclock-time) ~minutes
         !sensor has penetrated the sector with !(list sensor
         (ask !sensor recall your type) (car on-interval) (cadr on-interval)))
   )
)
```

```
(file-actors: sector)

(declare (*fexpr tell*))
(declare (specials t))

(ask simulator create generic sector with
          position        nil   ;((x1 y1) (x2 y2) (x3 y3) (x4 y4))
          weather-factor  nil   ;represents degree of good/bad weather
                                ;(1=excellent  .5=average etc.)
          terrain-factor  nil   ;represents degree of ease/difficulty of
                                ;sensing due to terrain irregularities
                                ;(1=flat,unimpeded  .3=hilly etc.)
          units           nil   ;a list consisting of sublists of
                                ;8 atoms ie (instance-id, unit-type, entrance-
                                ;time, entrance-position(x,y), exit-time,
                                ;exit-position(x,y))
          sensors         nil   ;a list consisting of sublists of 4 atoms
                                ;ie. (instance-id, sensor-type, on-time,
                                ;off-time)
          active-red-radars nil ;red radars currently turned on --
                                ;list of (unit id, type, freq, turn-on-time, turn-off-time)
          elint-sensors   nil   ;same format as sensors
)

(ask sector when receiving ()unit has penetrated sector with >unit-data)
   (~you add !unit-data to your list of units) ; update info in sector
   (~you plan after !(diff (caadddr unit-data) ~nclock-time)
       ~minutes remove !unit-data from your list of units)
   (prog (time-space-overlaps )
      (setq time-space-overlaps (ask mathematician determine time and space
                overlaps between !unit-data and any sensors in !myself))
      (if time-space-overlaps then   ; queue overlaps for sensor models
         (loop for sensor in time-space-overlaps
            do (tell !(car sensor) plan after !(diff (cadr sensor)
                ~nclock-time) ~minutes check for sensor detection of !unit)))))


(ask sector when receiving ()sensor has penetrated the sector with >sensor-data)
   (prog (time-space-overlaps)
      (if (or (equal (ask !sensor recall your type) 'mti-a)
              (equal (ask !sensor recall your type) 'mti-g)) then
         (~you add !sensor-data to your list of sensors) ; update info in sector
         (~you plan after !(diff (cadddr sensor-data) ~nclock-time)
             ~minutes remove !sensor-data from your list of sensors)
         (if (equal (ask !(car sensor-data) recall your type) 'mti-g) then
            (tell !sensor plan after !(diff (cadddr sensor-data)
               ~nclock-time) ~minutes set your units-found to nil)
         )
         (setq time-space-overlaps (ask mathematician determine time and space
               overlaps between !sensor-data and any units in !myself))
         (if time-space-overlaps then   ;queue overlaps for sensor models
            (loop for unit in time-space-overlaps
               do (tell !sensor plan after !(diff (cadr unit) ~nclock-time)
                   ~minutes check for sensor detection of !(car unit))))
       else  ; elint sensor has penetrated global sector
         (~you add !sensor-data to your list of elint-sensors)
         (~you plan after !(diff (cadddr sensor-data) ~nclock-time) ~minutes remove
             !sensor-data from your list of elint-sensors)
         (setq time-space-overlaps (ask mathematician determine time/space overlaps
              between elint !sensor-data and any red radars))
```

```
        (if time-space-overlaps then ; queue overlaps for formal sensor model
           (loop for radar-test in time-space-overlaps do
              (tell |sensor plan after |(diff (cadr radar-test) ~nclock-time)
                  ~minutes check for sensor detection of |(car radar-test)
                  of type |(caddr radar-test) and frequency |(cadddr radar-test))
              )
           )
        )
     )
  )


(ask sector when receiving ()unit has voice communication with frequency >freq
        and duration >duration)
   (prog (time-space-overlaps )
      (setq time-space-overlaps (ask mathematician determine time/space
         overlaps between |unit and any comint sensors))
      (if time-space-overlaps then   ; queue overlaps for formal sensor models
         (loop for sensor-test in time-space-overlaps do
            (tell |sensor-test check for sensor detection of |unit at frequency
                  |freq and duration |duration)
         )
      )
   )
)


(ask sector when receiving ()unit has fired >artillery-missile)
   (prog (time-space-overlaps)
      (setq time-space-overlaps (ask mathematician determine time/space overlaps
         between |unit and any cmcb sensors))
      (if time-space-overlaps then  ; queue overlaps for formal sensor models
         (loop for sensor-test in time-space-overlaps do
            (tell |sensor-test check for sensor detection of |unit)
         )
      )
   )
)


(ask sector when receiving ()unit has turned >type radar on at frequency >freq and duration >duration)
   (~you add |(list unit type freq ~nclock-time (plus ~nclock-time duration)) to your list of active-red-radars)
   (~you plan after |duration ~minutes remove |(list unit type freq ~nclock-time
     (plus ~nclock-time duration)) from your list of active-red-radars)
   (prog (time-space-overlaps)
      (setq time-space-overlaps (ask mathematician determine time/space overlaps
            between |unit on for |duration minutes and any elint sensors) )
      (if time-space-overlaps then
         (loop for sensor-test in time-space-overlaps do
            (tell |(car sensor-test) plan after |(diff (cadr sensor-test) ~nclock-time)
               ~minutes check for sensor detection of |unit of type |type and
               frequency |freq)
         )
      )
   )
)


(ask sector when receiving ()unit has turned >type radar off)
   nil
)
```

```
(ask sector when receiving ()sensor has taken picture at )point)
   (loop for unit-test in (units-within-picture sensor point) do
      (tell !sensor check for sensor detection of !unit-test)
   )
)
```

```
(file-actors: flightplanner)

(declare (*fexpr tell*))
(declare (specials t))

(ask simulator create instance flightplanner)


(ask flightplanner when receiving (move airborne sensor >sensor using
       racetrack pattern)

   ; convert the existing formatted plan to one which has the relevant
   ; template points of a holding pattern [there are 6 of these template
   ; points each having the form (t, (x,y)) ]

   (ask !sensor set your start-time to !(plus ~nclock-time
           (ask !sensor recall your response-delay)))

   (prog (time-at-first-pt time-at-second-pt time-at-third-pt time-at-fourth-pt
          time-at-fifth-pt time-at-sixth-pt first-pt second-pt third-pt
          fourth-pt fifth-pt sixth-pt internal-speed circuit-time leg-length
          leg-travel-time num-circuits new-plan hold-pat-radius n-circuit-time)

      (setq internal-speed (quotient (float (ask !sensor recall your speed))
                                     60.0))
      (setq leg-length (car (ask !sensor recall your plan)))
      (setq num-circuits (cadr (ask !sensor recall your plan)))

      ; first point (initial point -lower rt hand pt.)

      (setq first-pt (caddr (ask !sensor recall your plan)))
      (setq time-at-first-pt (plus (ask !sensor recall your start-time)
         (quotient (float (distance-between-2-pts (car (ask !sensor recall your
         beg-end-loc)) first-pt)) internal-speed)))
      (setq new-plan (list (list time-at-first-pt first-pt)))

      ; second point (top of right leg)

      (setq leg-travel-time (quotient (float leg-length) internal-speed))
      (setq time-at-second-pt (plus time-at-first-pt leg-travel-time))
      (setq second-pt (list (car first-pt) (plus (cadr first-pt) leg-length)))
      (setq new-plan (append new-plan (list (list time-at-second-pt
                             second-pt))))


      ; third point (midpt of turn)

      (setq time-at-third-pt (plus time-at-second-pt .5))
      (setq hold-pat-radius (times .318 internal-speed))
      (setq third-pt (list (diff (car second-pt) hold-pat-radius)
            (plus (cadr second-pt) hold-pat-radius)))
      (setq new-plan (append new-plan (list (list time-at-third-pt third-pt))))

      (if (member (ask !sensor recall your type) (ask sensor-action-unit recall
         your sector-sensors)) then
         (tell !sensor plan after !(diff time-at-third-pt ~nclock-time)
              ~minutes set your units-found to nil)
      )


      ; fourth point (top of left leg)

      (setq time-at-fourth-pt (plus time-at-third-pt .5))
```

```
(setq fourth-pt (list (diff (car second-pt) (times 2. hold-pat-radius))
            (cadr second-pt)))
(setq new-plan (append new-plan (list (list time-at-fourth-pt
                                fourth-pt))))


; fifth point (bottom of left leg)

(setq time-at-fifth-pt (plus time-at-fourth-pt leg-travel-time))
(setq fifth-pt (list (car fourth-pt) (diff (cadr fourth-pt) leg-length)))
(setq new-plan (append new-plan (list (list time-at-fifth-pt fifth-pt))))


; sixth point (middle of bottom turn)

(setq time-at-sixth-pt (plus time-at-fifth-pt .5))
(setq sixth-pt (list (car third-pt) (diff (cadr third-pt)
   (plus leg-length (times 2. hold-pat-radius)))))
(setq new-plan (append new-plan (list (list time-at-sixth-pt sixth-pt))))


(if (member (ask !sensor recall your type) (ask sensor-action-unit
   recall your sector-sensors)) then
   (tell !sensor plan after !(diff time-at-sixth-pt ~nclock-time)
      ~minutes set your units-found to nil)
)


; replicate these points in the plan according to the number of circuits
; around the holding pattern

(setq circuit-time (plus 2. (times 2. leg-travel-time)))
(setq num-circuits (diff num-circuits 1.))
(loop as n from 1 to num-circuits do
   (setq n-circuit-time (times n circuit-time))
   (setq new-plan (append new-plan (list (list (plus time-at-first-pt
     n-circuit-time) first-pt))))
   (setq new-plan (append new-plan (list (list (plus time-at-second-pt
     n-circuit-time) second-pt))))
   (setq new-plan (append new-plan (list (list (plus time-at-third-pt
     n-circuit-time) third-pt))))
   (if (member (ask !sensor recall your type)(ask sensor-action-unit
      recall your sector-sensors)) then
     (tell !sensor plan after !(diff (plus time-at-third-pt
      n-circuit-time) ~nclock-time) ~minutes set your units-found to nil)
   )
   (setq new-plan (append new-plan (list (list (plus time-at-fourth-pt
     n-circuit-time) fourth-pt))))
   (setq new-plan (append new-plan (list (list (plus time-at-fifth-pt
     n-circuit-time) fifth-pt))))
   (setq new-plan (append new-plan (list (list (plus time-at-sixth-pt
     n-circuit-time) sixth-pt))))
   (if (member (ask !sensor recall your type)(ask sensor-action-unit
      recall your sector-sensors)) then
     (tell !sensor plan after !(diff (plus time-at-sixth-pt
      n-circuit-time) ~nclock-time) ~minutes set your units-found to nil)
   )
)

; remove last sixth pt from new-plan
(setq new-plan (reverse (cdr (reverse new-plan))))

(ask !sensor set your plan to !new-plan)
```

```
;find time at final point

(ask |sensor set your stop-time to |(plus (caar (reverse new-plan))
  (quotient (float (distance-between-2-pts
    (cadr (ask |sensor recall your beg-end-loc))
      (cadar (reverse new-plan)))) internal-speed)))

; find update points for graphics

(ask mathematician calculate update points for |sensor)

    )
  )

(ask flightplanner when receiving (move airborne imint sensor >sensor along points >points)
   (ask |sensor set your start-time to |(plus ~nclock-time (ask |sensor recall your response-delay)))
   (prog (time-at-point plan internal-speed prev-pt prev-time leg-travel-time onoff-times)
      (setq internal-speed (quotient (float (ask |sensor recall your speed)) 60.0))
      (setq prev-time (ask |sensor recall your start-time))
      (setq prev-pt (car (ask |sensor recall your beg-end-loc)))

      (loop for point in points do
         (setq leg-travel-time (quotient (float (distance-between-2-pts prev-pt point)) internal-speed))
         (setq time-at-point (plus prev-time leg-travel-time))
         (ask |sensor plan after |(diff time-at-point ~nclock-time) ~minutes
                  set your current-status to on)
         (ask |sensor plan after |(diff time-at-point ~nclock-time) ~minutes
                  fly to |point)
         (ask sector plan after |(diff (plus time-at-point .0000001) ~nclock-time)
                 ~minutes |sensor has taken picture at |point)
         (setq plan (append plan (list (list time-at-point point))))
         (setq prev-pt point)
         (setq prev-time time-at-point)
      )
      (ask |sensor set your plan to |plan)
      (ask |sensor set your stop-time to |(plus (caar (reverse plan))
         (quotient (float (distance-between-2-pts (cadr (ask |sensor recall
         your beg-end-loc)) (cadar (reverse plan)))) internal-speed)))
      (ask mathematician calculate update points for |sensor)
   )
)
```

```
; Pathfinder determines the path a unit or ground based sensor (not yet turned
; on) will take.  It then asks the units and sensors to be at the specified
; nodes at the specified times and also asks mathematician to determine the
; penetration of sectors.

(file-actors: pathfinder)

(declare (*fexpr tell*))
(declare (specials t))

(ask simulator create instance pathfinder)

(ask pathfinder when receiving (move >unit from >point-a to >point-b)
   ; This section determines nodal pathways and times
   (if (not (ask !unit recall your plan))  then
      (if (not (ask !unit recall your start-time)) then
         (ask !unit set your start-time to !(plus ~nclock-time
              (ask !unit recall your response-delay)))
      )
      (ask !unit set your beg-end-loc to !(list point-a point-b))
      (get-route unit)
      (get-plan unit)
   )


   ; determine sector penetration of unit

   (ask mathematician determine sector penetration for !unit)


   ; plan movement through nodes

   (ask pathfinder-assistant plan after !(diff (ask !unit recall your start-time) ~nclock-time
      .1) ~minutes plan to move !unit) ;spaces out calcs.
)


(ask pathfinder when receiving (move ground sensor >sensor from >point-a to
                                >point-b)
   (if (not (ask !sensor recall your plan)) then
      (ask !sensor set your beg-end-loc to !(list point-a point-b))
      (ask !sensor set your start-time to !(plus ~nclock-time
           (ask !sensor recall your response-delay)))
      (get-route sensor)
      (get-plan sensor)
   )



   ;plan movement through nodes

   (ask pathfinder-assistant plan to move !sensor)
)
```

```
; pathfinder-assistant plans the moves of the objects for graphics updates

(file-actors: pathfinder-assistant)

(declare (*fexpr tell*))
(declare (specials t))

(ask simulator create instance pathfinder-assistant)

(ask pathfinder-assistant when receiving (plan to move >object)
   (prog (temp-plan)

     (if (member object (ask sensor recall your descendants)) then

      (loop for node-test in (ask !object recall your plan) do

         (setq temp-plan (append temp-plan (list (list (car node-test)
                 (ask !(cadr node-test) recall your position)))))
      )

      else  ; object is a unit -- send node arrival messages

      (loop for node-test in (ask !object recall your plan) do

         ; notify unit it has reached a node

         (ask !object plan after !(diff (car node-test) ~nclock-time)
            ~minutes have arrived at node !(cadr node-test))

         (setq temp-plan (append temp-plan (list (list (car node-test)
              (ask !(cadr node-test) recall your position)))))
      )
     )

     (ask !object set your plan to !temp-plan)
     (ask mathematician calculate update points for !object)

     ; send closing communications at final point on route

     (if (not (member object (ask sensor recall your descendants))) then

      (ask !object plan after !(diff (ask !object recall your stop-time)
         ~nclock-time) ~minutes have arrived at final destination)
     )
   )
)
```

```
(file-actors: node)

(ask simulator create generic node with
        position              nil
        sector                nil
        type                  nil
        number-linked-nodes   nil
        linked-nodes          nil
)
```

```
; Interface is a ROSS object which will act as the interface between BEM
; and the BLUE SENSOR CONTROL STATION and ANALYST. Notification of all
; sensor detections will be sent to interface for forwarding.

(file-actors: interface)

(declare (specials t))
(sstatus uctolc nil)

(ask simulator create instance interface)

(ask interface when receiving (mti >sensor has detected >unit)
   (if (ask simulator recall your blue-sensor-control-station-status) then
      (prog (sensor-report)
         (setq sensor-report (list
            'mover
            'sensor-id  sensor
            'position (position-at-time-t unit ~nclock-time)
            'sensetime (minutes-to-scores ~nclock-time)
            'reporttime (minutes-to-scores ~nclock-time)
            'xerror '0.1
            'yerror '0.1
            'type 'track
            'speed (ask !unit recall your night-speed) ; for the moment
            'track-length (ask !unit recall your night-length) ; for the moment
            'direction (direction unit ~nclock-time)  )
         )
         (ask sensor-action-unit add !sensor-report to your list of
             stored-sensor-reports)
      )
   )
   (if (ask simulator recall your analyst-status) then
      (prog (analyst-report position)
         (setq position (position-at-time-t unit ~nclock-time))
         (setq analyst-report (list
            'TYPE 'MOVER
            'SENSOR-ID sensor
            'RTIME (minutes-to-scores ~nclock-time)
            'STIME (minutes-to-scores ~nclock-time)
            'X (car position)
            'Y (cadr position)
            'RX '0.1
            'RY '0.1
            'MTYPE 'TREAD
            'SPEED (ask !unit recall your night-speed) ;for the moment
            'LEN (ask !unit recall your night-length)  ;for the moment
            'DIR (direction unit ~nclock-time) )
         )
         (ask sensor-action-unit add !analyst-report to your list of stored-analyst-reports)
      )
   )
)


(ask interface when receiving (comint >sensor has detected >unit at
                              frequency >freq and duration >duration)
   (if (ask simulator recall your blue-sensor-control-station-status) then
      (prog (sensor-report)
         (setq sensor-report (list
            'comint
            'sensor-id sensor
            'frequency freq
            'position (position-at-time-t unit ~nclock-time)
```

```
                'sensetime (minutes-to-scores ~nclock-time)
                'reporttime (minutes-to-scores ~nclock-time)
                'xerror '0.1
                'yerror '0.1
                'transmission-length duration )
            )
            (ask sensor-action-unit add !sensor-report to your list of
                stored-sensor-reports)
        )
    )
    (if (ask simulator recall your analyst-status) then
        (prog (analyst-report position)
            (setq position (position-at-time-t unit ~nclock-time))
            (setq analyst-report (list
                'TYPE 'COMINT
                'SENSOR-ID sensor
                'RTIME (minutes-to-scores ~nclock-time)
                'STIME (minutes-to-scores ~nclock-time)
                'X (car position)
                'Y (cadr position)
                'RX '0.1
                'RY '0.1
                'FREQ freq
                'LTRANS duration  )
            )
            (ask sensor-action-unit add !analyst-report to your list of stored-analyst-reports)
        )
    )
)


(ask interface when receiving (cmcb )sensor has detected )unit)
    (if (ask simulator recall your blue-sensor-control-station-status) then
        (prog (sensor-report)
            (setq sensor-report (list
                'shooter
                'sensor-id sensor
                'position (position-at-time-t unit ~nclock-time)
                'sensetime (minutes-to-scores ~nclock-time)
                'reporttime (minutes-to-scores ~nclock-time)
                'xerror '0.1
                'yerror '0.1
                'type (get-burst-type unit) )
            )
            (ask sensor-action-unit add !sensor-report to your list of stored-sensor-reports)
        )
    )
    (if (ask simulator recall your analyst-status) then
        (prog (analyst-report position)
            (setq position (position-at-time-t unit ~nclock-time))
            (setq analyst-report (list
                'TYPE 'SHOOTER
                'SENSOR-ID sensor
                'RTIME (minutes-to-scores ~nclock-time)
                'STIME (minutes-to-scores ~nclock-time)
                'X (car position)
                'Y (cadr position)
                'RX '0.1
                'RY '0.1
                'STYPE (get-burst-type unit) )
            )
            (ask sensor-action-unit add !analyst-report to your list of stored-analyst-reports)
```

```
      )
    )
  )

  (ask interface when receiving (elint >sensor has detected >unit of type >type
     and frequency >freq)
     (if (ask simulator recall your blue-sensor-control-station-status) then
        (prog (sensor-report)
           (setq sensor-report (list
              'radar
              'sensor-id sensor
              'position (position-at-time-t unit ~nclock-time)
              'sensetime (minutes-to-scores ~nclock-time)
              'reporttime (minutes-to-scores ~nclock-time)
              'xerror '0.1
              'yerror '0.1
              'frequency freq
              'type type )
           )
           (ask sensor-action-unit add !sensor-report to your list of stored-sensor-reports)
        )
     )
     (if (ask simulator recall your analyst-status) then
        (prog (analyst-report position)
           (setq position (position-at-time-t unit ~nclock-time))
           (setq analyst-report (list
              'TYPE 'RADAR
              'SENSOR-ID sensor
              'RTIME (minutes-to-scores ~nclock-time)
              'STIME (minutes-to-scores ~nclock-time)
              'X (car position)
              'Y (cadr position)
              'RX '0.1
              'RY '0.1
              'FREQ freq
              'PRF (cadr (memq 'prf (ask !unit recall your !type)))
              'PW (cadr (memq 'pw (ask !unit recall your !type)))  )
           )
           (ask sensor-action-unit add !analyst-report to your list of stored-analyst-reports)
        )
     )
  )

  (ask interface when receiving (imint >sensor has detected >unit at time >time)
     (if (ask simulator recall your blue-sensor-control-station-status) then
        (prog (sensor-report)
           (setq sensor-report (list
              'pi
              'sensor-id sensor
              'position (position-at-time-t unit time)
              'sensetime (minutes-to-scores time)
              'reporttime (minutes-to-scores  ~nclock-time)
              'xerror '0.1
              'yerror '0.1
              'niirsnum (ask !unit recall your niirsnum)
              'type 'photo )
           )
           (ask sensor-action-unit add !sensor-report to your list of stored-sensor-reports)
        )
     )
     (if (ask simulator recall your analyst-status) then
```

```
(prog (analyst-report position)
   (setq position (position-at-time-t unit time))
   (setq analyst-report (list
      'TYPE 'PI
      'SENSOR-ID sensor
      'RTIME (minutes-to-scores ~nclock-time)
      'STIME (minutes-to-scores time)
      'X (car position)
      'Y (cadr position)
      'RX '0.1
      'RY '0.1
      'OBJNO (ask !unit recall your niirsnum) )
   )
   (ask sensor-action-unit add !analyst-report to your list of stored-analyst-reports)
  )
 )
)
```

```
;   scheduler -- actor to schedule artillery firings and radar emissions

(file-actors: scheduler)

(declare (*fexpr tell*))
(declare (specials t))


(ask simulator create instance scheduler  with
  prep-start-time     nil     ;  start of artillery preparation
  prep-stop-time      nil     ;  stop time of artillery preparation
  prep-length         nil     ;  length of artillery preparation
  firing-units        nil     ;  firing units available for various missions
  prep-freq           .5      ;   firing frequency during preparation
  fire-mission-freq   .5      ;   firing frequency during battle
  )

(ask scheduler when receiving (cease call for fire)

  (~you unplan all (initiate call for fire)))


(ask scheduler when receiving (initiate call for fire)

  (~block with local variables (candidates)

                  ;first schedule next call

                  (~you plan after !(expon-dist (~your fire-mission-freq)) ~minutes
                                initiate call for fire)

                  (loop for unit in (append (~every mr-co) (~every tank-co))

                                do      (cond  ((equal (get unit 'status)  'contact)

                                                        (setq candidates (cons unit candidates)))))

                  (cond ((null candidates) (return nil))
                                ;  now have list of units, pick one to fire

                                (t (ask !(pick-unit candidates) call for fire)))))


 (ask scheduler when receiving

          (conduct >t-length minutes fire preparation

                                starting at time >t-start)

          (~you set your prep-start-time to !t-start)

          (~you set your prep-length to !t-length)

          (~you set your prep-stop-time to !(add t-start t-length))

          (~you set your firing-units to
                !(append (~every arty-btry) (~every gunhow-btry) (~every gun-btry)))

          (~you fire the prep))
```

```
(ask scheduler when receiving (cease fire the prep)

  (~you unplan all (fire the prep)))


(ask scheduler when receiving (initiate red radars)

  (~block with local variables (unit)

                (~you plan after ((expon-dist 15) ~minutes
                              create sa-6 acquisition)

                (loop for unit in
                              (append (~every mr-bn-cop)
                                      (~every tank-bn-cop))

                      do      (ask scheduler-assistant schedule duty cycle for
                                      ~the unit firecontrol radar))

                (loop for unit in (~every sa-6-bn-cp)

                      do      (ask scheduler-assistant schedule duty cycle for
                                      ~the unit surveillance radar))

                (loop for unit in (~every tgt-acq-btry)

                      do      (ask scheduler-assistant schedule duty cycle for
                                      ~the unit met radar)

                              (ask scheduler-assistant schedule duty cycle for
                                      ~the unit surveillance radar))

                (loop for unit in
                              (append (~every arty-bn-cp)
                                              (~every gunhow-bn-cp)
                                              (~every gun-bn-cp))

                      do      (ask scheduler-assistant schedule duty cycle for
                                      ~the unit surveillance radar))))


(ask scheduler when receiving
  (create sa-6 acquisition)

  (~you plan after ((expon-dist 15) ~minutes
                create sa-6 acquisition)

  (loop for unit in (~every sa-6-bn-cp)
                ;  check status of rdrsurv-4  (times 2)

                when  (equal  (get-radar-status unit 'surveillance) 'on)
                          return (tell ~the unit target acquired)))


(ask scheduler when receiving
  (cease sa-6 acquisition)

  (~you unplan all (create sa-6 acquisition)))
```

```
(ask scheduler when receiving (fire the prep)
  (~block with local variables (firing-unit unit-activity)
          (if (greaterp (ask nclock recall your $stime)
                                    (~your prep-stop-time))
              then (return  nil)
              else (~you plan after !(expon-dist (~your prep-freq))
                                    ~minutes fire the prep))
          (loop for firing-unit in (~your firing-units)
                when (and
                          (not (equal (setq unit-activity
                                                         (get-unit-activity firing-unit))
                                                   'moving))
                          (not (equal unit-activity  'firing)))
                do  (~you set your firing-units to
                                  ! (append1 (delete firing-unit
                                                          (~your firing-units)) firing-unit))
                (return (ask !firing-unit fire prep fires)))))
```

```
(file-actors: scheduler-assistant)
(declare (*fexpr tell*))
(declare (specials t))

(ask simulator create instance scheduler-assistant)

(ask scheduler-assistant when receiving

  (schedule duty cycle for >unit >type radar)

  (~block with local variables (start-time)

                (~let start-time be
                              (times 5. (int-urngen 12)))

                (ask ~the unit plan after ~the start-time ~minutes
                        turn ~the type radar on)

                (ask ~the unit plan after !(sum start-time 15) ~minutes
                        turn ~the type radar off)

                ( ask scheduler-assistant plan after 75 ~minutes schedule duty cycle for
                        ~the unit !type radar)

                (return start-time)))
```

```
;**************************************************************************
;                     THE ARTIST
;**************************************************************************
; begun                03/08/83 jrd
; updated              09/02/83 jrd
;**************************************************************************
; files required:
; ----code
;       artist.l              graphics interface with ross simulation
;       artfns.l              artist support fns in lisp
;       aed.l                 aed512 lisp-based driver
; ----data
;       colorN.l              color lookup table additions
;       fontN.l               font character set
;**************************************************************************
; globals to the artist:
; background black
; textplane textcolor gridcolor
; roadcolor towncolor bridgecolor obstaclecolor
; townfont bridgefont
; terrain-loaded
;**************************************************************************
;                           ross creation
;**************************************************************************
(file-actors: artist)
(declare (specials t))
(ask something create generic artist with
       arc-increment   12.5
       terrain-loaded  nil)
;**************************************************************************
;                     artist controls aed512's
;**************************************************************************

(ask artist when receiving (send display to >terminal)         ;port to aed
                (go-remote terminal)]

(ask artist when receiving (begin painting)                    ;setup aed
                (start-aed)(init-aed)
                (setq $ldprint nil)
                (define-fonts)  (define-colors)
                (setq $ldprint t)
                (stop-aed)]

(ask artist when receiving (set window to >range)              ;scale aed
                (set-window range)]

;**************************************************************************
;       artist interaction with simulation entities
;**************************************************************************

(ask artist when receiving (correspond)
   (prog (ans loc searchlist item foo header unit-list sensor-list node-list)
        ;units
            (setq unit-list (remove-nils
                (mapcar 'check-if-instance (ask unit recall your instances))))
        ;sensors
            (setq sensor-list (remove-nils
        (mapcar 'check-if-instance (ask sensor recall your instances))))
        ;nodes
            (setq node-list (get 'node 'offspring))
        (setq header '**********************************************)
                loop
                (print header)(terpr)
```

```
                (patom "U(nit S(ensor N(ode Z(ector Q(uit ?")(terpr)
                (setq ans (read))
                (and (not (memq ans '(u s n z q)))(go loop))
                (and loc                ; if loc non-nil erase circle
                        (start-aed)
                        (move-to loc)
                        (erase-circle 15)
                        (stop-aed))
                (and (eq ans 'q)(return t))
                (setq loc (getpick))
    (cond
        ((eq ans 'u)
                (setq item (search-location unit-list loc)))
        ((eq ans 's)
                (setq item (search-location sensor-list loc)))
        ((eq ans 'n)
                (setq item (search-location node-list loc)))
        (t
                (setq item (find-sector loc))
                (cond ((null item)(patom "item not found at ")(patom loc)
                                (terpr)(go loop)))
                (go display-data)))

                (cond ((null item)(patom "item not found at ")(patom loc)
                                (terpr)(go loop)))
                (setq loc (get item 'position))
                (start-aed)
                (move-to loc)
                (go-textplane)
                (draw-circle 15)
                (stop-aed)
                display-data
                        (print header)(terpr)
                        (print item)(terpr)
                        (print header)(terpr)
                        ($prpr (plist item))(terpr)
                        (go loop)))

;***********************************************************************
;                       background picture routines
;***********************************************************************

(ask artist when receiving ()? grid marks)
                (cond
                     ((memq ? '(draw show display))
                                        (show-grid gridcolor))
                        ((eq ? 'erase)(hide-grid gridcolor))
                        (t nil))]

(ask artist when receiving ()? tic marks)
                (cond
                     ((memq ? '(draw show display))
                                        (show-tics gridcolor))
                        ((eq ? 'erase)(hide-tics gridcolor))
                        (t nil))]

(ask artist when receiving ()? sectors)
        (prog (poslist slist s one two three four )
        (start-aed)
        (set-write-mask textplane)
        (cond
                     ((memq ? '(draw show display))
                                (set-color textcolor))
```

```
                ((eq ? 'erase)(set-color black))
                (t nil))
        (setq slist (get 'sector 'offspring))
        (mapcar 'draw-sector slist)
        (stop-aed)]

(ask artist when receiving (>? node network)
        (prog (linecolor nodecolor)
                (start-aed)
                (set-write-mask background)
                (setq linecolor gridcolor)
                (setq nlist (get 'node 'offspring))
                loop1
                        (cond
                                ((null (car nlist)) (go endit)))
                        (setq typ (get (car nlist) 'type))
                        (cond
                                ((eq typ 'normal)
                                 (setq nodecolor black))
                                ((eq typ 'town)
                                 (setq nodecolor towncolor))
                                ((eq typ 'city)
                                 (setq nodecolor towncolor))
                                ((eq typ 'bridge)
                                 (setq nodecolor bridgecolor))
                                ((eq typ 'obstacle)
                                 (setq nodecolor obstaclecolor))
                                (t nil))
                (cond
                     ((memq ? '(draw show display))
                                        nil)
                        ((eq ? 'erase)
                                (setq nodecolor black)
                                (setq linecolor black))
                        (t nil))
                (set-color nodecolor)
        (move-to (get (car nlist) 'position))
        (draw-circle 3)
        (setq llist (get (car nlist) 'linked-nodes))
                loop2
                        (set-color linecolor)
                        (draw-clipped-line
                                 (get (car nlist) 'position)
                                 (get (car llist) 'position))
                        (setq llist (cdr llist))
                        (cond
                                ((eq (car llist) nil)
                                        (setq nlist (cdr nlist))
                                        (go loop1))
                                (t (go loop2)))
              endit
                (go-textplane)
                (stop-aed)]

(ask artist when receiving (>? towns and bridges)
        (prog (nodesymbol nlist)
                (start-aed)
                (set-write-mask background)
                (setq nlist (get 'node 'offspring))
        (cond
                ((memq ? '(draw show display))
                    (mapcar 'show-node nlist))
                ((eq ? 'erase)
```

```
                (mapcar 'hide-node nlist))
             (t nil))
       (go-textplane)
       (stop-aed)]

(ask artist when receiving ()? terrain features)
       (and (not (get 'artist 'terrain-loaded))
               (load-roads)(load-rivers)(load-woods)
               (putprop 'artist t 'terrain-loaded))
       (cond
             ((memq ? '(draw show display))
                         (display-roads)
                         (display-rivers)
                         (display-woods))
             ((eq ? 'erase)
                         (erase-roads)
                         (erase-rivers)
                         (erase-woods))
             (t nil))
       )
;*****************************************************************************
;                        graphical display routines
;*****************************************************************************


;******************************
;      shooting activity
;******************************

(ask artist when receiving ()? >type burst at >location)
    (prog (tint n)
       (and (not (within location))(return nil))
       (setq n 0)
       (start-aed)
       loop
       (setq n (add1 n))
       (cond
           ((eq type 'tube)(setq tint 1))
           ((eq type 'missile)(setq tint 8))
           ((eq type 'rocket)(setq tint 16))
           (t (setq tint 1)))
       (cond ((memq ? '(show place display draw))
                (set-write-mask tint)
                (move-to location)
                (move-pixel '(-12 -4))
                (set-color tint)
                (label '*)
                (cond
                      ((lessp n 4)(go loop))
                      (t (move-to location)
                         (move-pixel '(-12 -4))
                        (set-color 0)
                        (label '*)
                        (stop-aed))
                   )
                )
           (t nil]

;****************************
;      elint activity
;****************************

(ask artist when receiving ()? >type radar at >location)
```

```
        (prog (shape)
        (and (not (within location))(return nil))
        (start-aed)
        (move-to location)
        (move-pixel '(-8 0))
    (cond
        ((eq type 'firecontrol)(setq shape 30)(move-pixel '(0 4)))
        ((eq type 'heightfinder)(setq shape 30))
        ((eq type 'surveillance)(setq shape 31)(move-pixel '(0 4)))
        ((eq type 'met)(setq shape 32))
        (t nil))

    (set-write-mask (select-plane shape))

    (cond
             ((memq ? '(draw show display))
                        (put-font shape))
                ((eq ? 'erase)
                        (erase-area 8 10))
                (t nil))
        (stop-aed)]

;****************************
;       clock
;****************************

(ask artist when receiving (update clock to >time)              ;display clock
                (start-aed)(go-home)(go-textplane)
                (erase-area 100 10)(exit-interpreter)
                (patom " CLOCK: ")(patom time)(terpr)
                (escape)(stop-aed)]

;********************************
;       movement activity
;********************************

(ask artist when receiving (>? >shape at >vicinity)
        (prog ()
                (and (not (within vicinity))(return nil))
                (start-aed)
                (move-to vicinity)
                (move-pixel '(-4 -5))
                (set-write-mask (select-plane shape))
        (cond
             ((memq ? '(place draw show display))
                        (put-font shape))
                ((eq ? 'erase)
                        (erase-area 8 10))
                (t nil))
        (stop-aed)]

;******************************
;       comint activity
;******************************

(ask artist when receiving (>? >type comm at >location)
                (prog (tint)
                (and (not (within location))(return nil))
                (cond
                     ((not (memq ? '(draw show display)))
                        (return nil)))
                (cond
```

```
                ((eq type 'firecall)(setq tint 1))
                ((eq type 'netcall)(setq tint 16))
                ((eq type 'readycall)(setq tint 4))
                ((eq type 'controlcall)(setq tint 2))
                ((eq type 'sitrepcall)(setq tint 2))
                ((eq type 'confirmcall)(setq tint 8))
                (t nil))
        (start-aed)
        (move-to location)
        (radiate tint)
        (stop-aed)]

;************************************
;       sensor activity
;************************************
(ask artist when receiving ()? >actor sensor coverage)
    (prog (range thetad location width length)
        (setq range (getn actor 'range)
              location (getn actor 'position)
              thetad (getn actor 'coverage-theta)
                width (getn actor 'coverage-width)
                length (getn actor 'coverage-length))
        (and (null thetad)(setq thetad 0))
        (and (not (evenp thetad)) (setq thetad (add1 thetad)))
        (start-aed)
        (set-write-mask textplane)
        (cond
              ((memq ? '(draw show display))
                (cond
                   ((equal thetad 360.) ; circle
                        (putprop actor (calc-circle-pts range location
                                (get 'artist 'arc-increment))
                                  'coverage-pts))
                   ((not (null width))  ; rectangle/square
                        (putprop actor (calc-box-pts location width length)
                                          'coverage-pts))

                   (t                   ; fan
                        (putprop actor (calc-fan-pts range location thetad
                                (get 'artist 'arc-increment))
                                  'coverage-pts)))
                (set-color textcolor)
                (draw-clipped-polygon (get actor 'coverage-pts)))

              ((eq ? 'erase)
                        (set-color black)
                        (draw-clipped-polygon (get actor 'coverage-pts))
                        (putprop actor nil 'coverage-pts)))
        (stop-aed)))


;**************************
;       movement activity
;**************************

(ask artist when receiving (move >me from >location to >destination
        using >shape)
                (start-aed)
                (set-color (select-plane shape))
```

```
(move-pixel '(4 5))
(draw-to destination)
(move-to location)
(erase-to destination)
(move-pixel '(-4 -5))
(put-font shape)
(stop-aed)]
```

```
(file-actors: dead-artist)

(ask simulator create generic dead-artist)

(ask dead-artist when receiving (+) nil)
```

```
; sensor instance file for tank division demo

(file-actors: controll comintl comint2 comint3 comint4 mtil mti2 mti3 mti4
              cmcbl elintl elint2 imintl)


(ask sensor-control-unit create instance controll with
   beg-end-loc   ((45 19) (45 19))
   subordinates (comintl comint2 comint3 comint4 mtil mti2 mti3 mti4 cmcbl
                 elintl elint2 imintl)
)


(ask comint create instance comintl with
   range  5
   beg-end-loc  ( (42 20) nil )
   type comint-g
   shape  42
   freq-range (10 50)
   night-speed 90
)

(ask comint create instance comint2 with
   range  5
   beg-end-loc  ( (42 21) nil )
   type comint-g
   shape  42
   freq-range (10 50)
   night-speed 90
)

(ask comint create instance comint3 with
   range  8.9
   beg-end-loc ( (48 19) (48 19) )
   type comint-a
   shape  41
   freq-range (10 50)
   speed  333  ; km/hr
)

(ask comint create instance comint4 with
   range  8.9
   beg-end-loc ( (48 19) (48 19) )
   type comint-a
   shape  41
   freq-range  (10 50)
   speed 333  ; km/hr
)

(ask mti create instance mtil with
   range    5    ; km
   beg-end-loc  ( (42.5 28.0) nil )
   type  mti-g
   shape  42
   coverage-theta 45
   night-speed 90
)

(ask mti create instance mti2 with
   range    5    ; km
   beg-end-loc  ( (44 22) nil )
   type mti-g
   shape 42
```

```
   coverage-theta 45
   night-speed 90
)

(ask mti create instance mti3 with
   range   25
   beg-end-loc  ( (48 19) (48 19) )
   type mti-a
   shape  41
   coverage-theta 45
   speed  333  ; km/hr
)

(ask mti create instance mti4 with
   range  25
   beg-end-loc (  (48 19) (48 19) )
   type  mti-a
   shape  41
   coverage-theta  45
   speed  333  ; km/hr
)

(ask cmcb create instance cmcb1 with
   range 9
   beg-end-loc ( (44 23) nil )
   type cmcb-g
   shape 42
   coverage-theta 60
   night-speed 90
)

(ask elint create instance elint1 with
   range 9
   beg-end-loc ( (44 24) nil,
   type elint-g
   shape 42
   coverage-theta 30
   night-speed 90
)

(ask elint create instance elint2 with
   range 25
   beg-end-loc ( (48 19) (48 19) )
   type elint-a
   shape 41
   coverage-theta 30
   speed 333  ; km/hr
)

(ask imint create instance imint1 with
   coverage-length  5  ;east-west
   coverage-width   5  ;north-south
   beg-end-loc ( (48 19) (48 19) )
   type imint-a
   shape 41
   speed 333 ; km/hr
)
```

```
; (divdemo2) scenario for the tank division

(ask simulator create instance divdemo2 with
   aed                        'txa2;
   simulation-duration        600
   sim-to-real-time-ratio     1000000
   window                     (41.0 78.0 18.0 41.0)  ; set for graphics
   graphics-scene             terrain
   ticksize                   .1
)

(setq slice-time 330)

(ask divdemo2 when receiving (prepare to start simulation)

;correction for red instance data
(ask tank-div-fwd16 add sa-6-bn-cp10 to your list of subordinates)

(ask imint set your reporting-delay to 0)


   (tell tank-div-fwd16 implement battle plan)
   (tell scheduler plan after 2 ~minutes initiate call for fire)
   (tell scheduler plan after 2 ~minutes initiate red radars)



   (tell controll plan after 3 ~minutes send ground mti sensor to setup
     point (51. 27.) for 100 minute mission)

   (tell controll plan after 3 ~minutes send ground comint sensor to setup
     point (54 25) for 60 minute mission)

   (tell controll plan after 3 ~minutes send ground cmcb sensor to setup
     point (53 23) for 100 minute mission)

   (tell controll plan after 3 ~minutes send ground elint sensor
           to setup point (52 24) for 100 minute mission)


   (tell controll plan after 3 ~minutes send up airborne comint sensor in
     5 km racetrack pattern 3 times around with initial pt (55 27))

   (tell controll plan after 3.3 ~minutes send up airborne mti sensor in
     12 km racetrack pattern 3 times around with initial pt (50 25) )

   (tell controll plan after 3.6 ~minutes send up airborne elint sensor in 8 km
      racetrack pattern 3 times around with initial pt (51.5 24) )

   (tell controll plan after 3.9 ~minutes send up airborne imint sensor to
      points ( (54 21) (69 27) (63 31) (54 27) ) )

;   (second wave of airborne)

   (tell controll plan after 30 ~minutes send up airborne comint sensor in
     5 km racetrack pattern 3 times around with initial pt (55 27))

   (tell controll plan after 30.3 ~minutes send up airborne mti sensor in
     12 km racetrack pattern 3 times around with initial pt (50 25) )

   (tell controll plan after 30.6 ~minutes send up airborne elint sensor in 8 km
      racetrack pattern 3 times around with initial pt (51.5 24) )
```

```
  (tell controll plan after 30.9 ~minutes send up airborne imint sensor to
     points ( (54 21) (69 27) (63 31) (54 27) ) )

; (third airborne wave)

  (tell controll plan after 57 ~minutes send up airborne comint sensor in
    5 km racetrack pattern 3 times around with initial pt (55 27))

  (tell controll plan after 57.3 ~minutes send up airborne mti sensor in
    12 km racetrack pattern 3 times around with initial pt (50 25) )

  (tell controll plan after 57.6 ~minutes send up airborne elint sensor in 8 km
     racetrack pattern 3 times around with initial pt (51.5 24) )

  (tell controll plan after 57.9 ~minutes send up airborne imint sensor to
     points ( (54 21) (69 27) (63 31) (54 27) ) )

;  fourth airborne wave

  (tell controll plan after 84 ~minutes send up airborne comint sensor in
    5 km racetrack pattern 3 times around with initial pt (55 27))

  (tell controll plan after 84.3 ~minutes send up airborne mti sensor in
    12 km racetrack pattern 3 times around with initial pt (50 25) )

  (tell controll plan after 84.6 ~minutes send up airborne elint sensor in 8 km
     racetrack pattern 3 times around with initial pt (51.5 24) )

  (tell controll plan after 84.9 ~minutes send up airborne imint sensor to
     points ( (54 21) (69 27) (63 31) (54 27) ) )

;  fifth airborne wave

  (tell controll plan after 111 ~minutes send up airborne comint sensor in
    5 km racetrack pattern 3 times around with initial pt (55 27))

  (tell controll plan after 111.3 ~minutes send up airborne mti sensor in
    12 km racetrack pattern 3 times around with initial pt (50 25) )

  (tell controll plan after 111.6 ~minutes send up airborne elint sensor in 8 km
     racetrack pattern 3 times around with initial pt (51.5 24) )

  (tell controll plan after 111.9 ~minutes send up airborne imint sensor to
     points ( (54 21) (69 27) (63 31) (54 27) ) )

;  sixth airborne wave

  (tell controll plan after 138 ~minutes send up airborne comint sensor in
    5 km racetrack pattern 3 times around with initial pt (55 27))

  (tell controll plan after 138.3 ~minutes send up airborne mti sensor in
    12 km racetrack pattern 3 times around with initial pt (50 25) )

  (tell controll plan after 138.6 ~minutes send up airborne elint sensor in 8 km
     racetrack pattern 3 times around with initial pt (51.5 24) )

  (tell controll plan after 138.9 ~minutes send up airborne imint sensor to
     points ( (54 21) (69 27) (63 31) (54 27) ) )
```

APPENDIX V

DEVELOPMENT/DEMONSTRATION FACILITIES

TABLE OF CONTENTS

1.0 DEVELOPMENT/DEMONSTRATION FACILITIES

Development of the BEM has taken place in the MITRE Washington Secure Processing Laboratory (SPL). Formal demonstration of a BEM simulation usually takes place in MITRE's Command and Management Information System (CAMIS) Laboratory.

1.1 SPL Facility

The SPL is located within a special compartmented information facility. Included in this facility are a VAX 11/780 with 8 megabytes of memory, one 67 megabyte disk drive, two 300 megabyte disk drives, and several graphics and text terminals. Also located in the SPL is an LMI LISP machine. Figure V-1 shows the configuration of the SPL. The BEM runs on the VAX 11/780. The ANALYST can run on both the VAX and the LISP machine.

1.2 CAMIS Facility

The CAMIS Lab is located next door to the SPL. It contains a VAX 11/780 configuration similar to that in the SPL. The CAMIS lab contains a briefing room where a graphics terminal display may be projected onto a large screen. A demonstration may be shown here by transferring the BEM simulation code by tape from the SPL to the CAMIS computer or by direct cable hookup to the SPL VAX. Figure V-2 shows the CAMIS Lab Configuration.

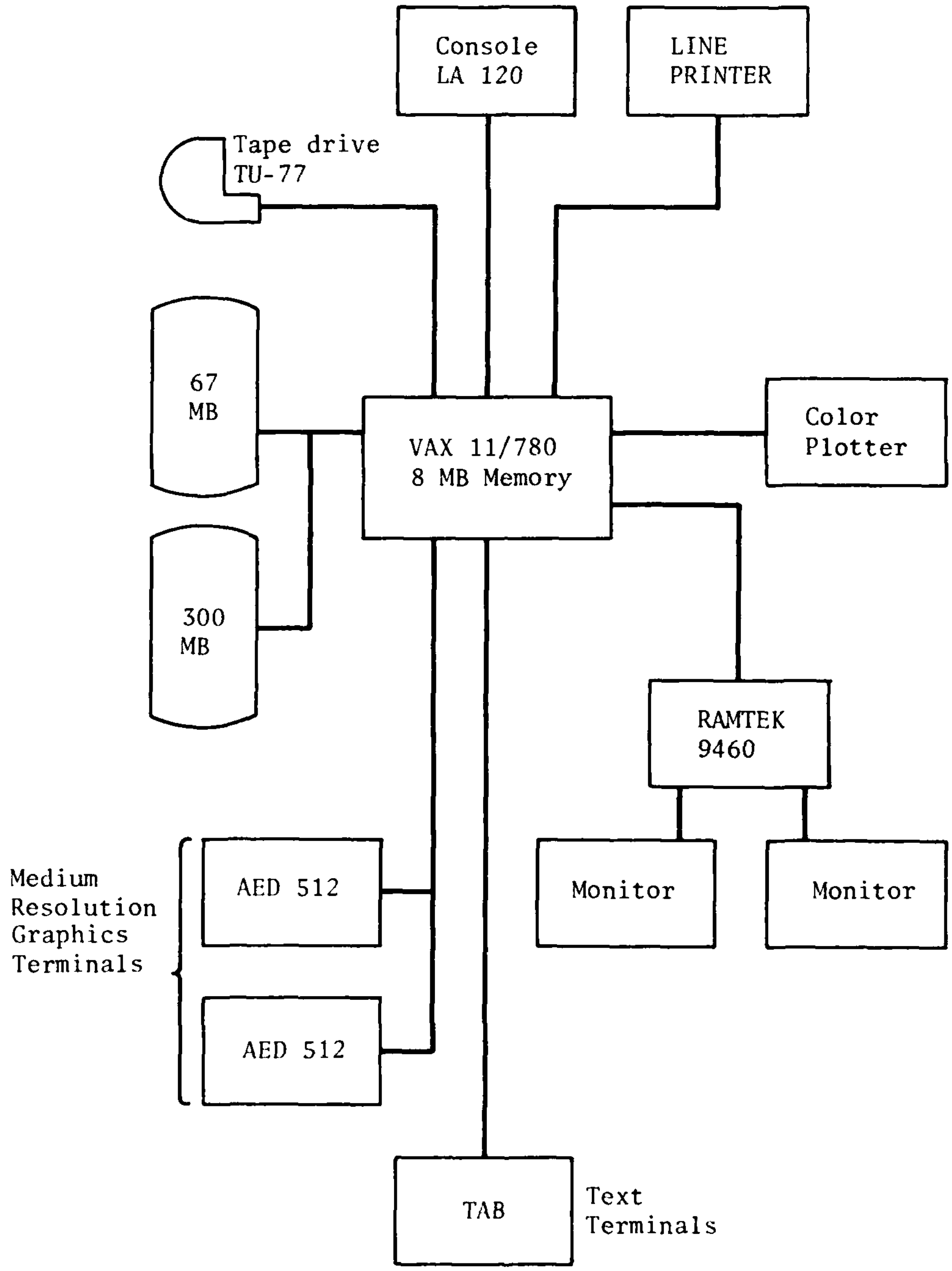
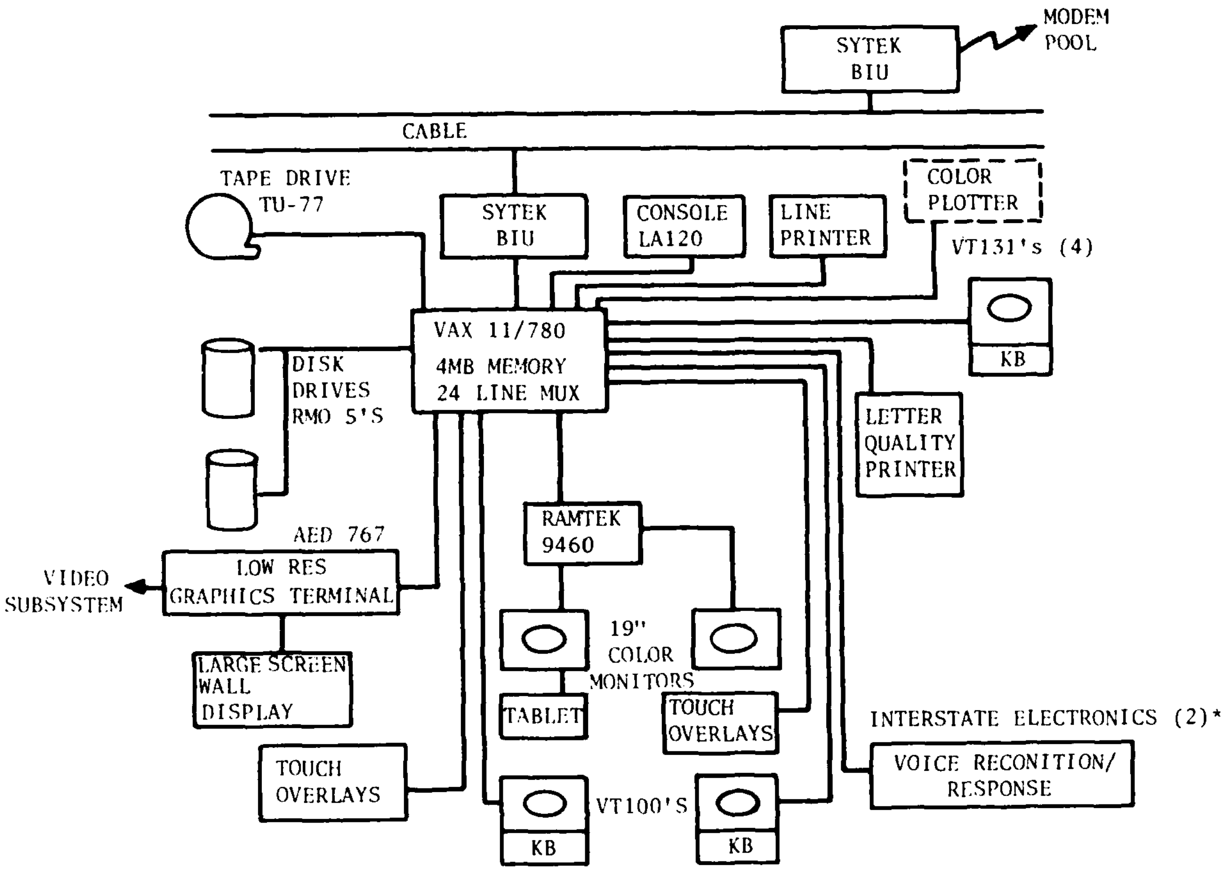


FIGURE V-1
SPL CONFIGURATION



* NOT CURRENTLY AVAILABLE

FIGURE V-2
CAMIS LABORATORY CONFIGURATION

GLOSSARY

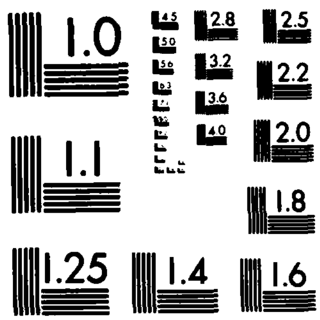
ACQ	Acquisition
ANALYST	An expert system for processing intelligence returns from the battlefield
ARTY	Artillery
ATGM	Anti-Tank Guided Missile
Bde	Brigade
BEM	Battlefield Environment Model
BN	Battalion
BSCS	Blue Sensor Control Station
BTRY	Battery
CAMIS	Command and Management Information System (Laboratory)
CMCB	Counter Mortar/Counter Battery Sensor
CO	Company
COMINT	Communications Intelligence Sensor
CP	Command Post
C^2	Command & Control
ELINT	Electronic Intelligence Sensor
FROG	Free Rocket Over Ground
FWD	Forward (CP)
GUN-HOW	Gun-Howitzer
IMINT	Imagery (Photo) Intelligence Sensor
LISP	List Processing computer language

GLOSSARY
(Concluded)

MAIN	Main (CP)
MR	Motorized Rifle
MRL	Multiple Rocket Launcher
MTCE	Maintenance
MTEG	MITRE Threat Event Generator
MTI	Moving Target Indicator Sensor
Pltn	Platoon
POL	Petroleum, Oil, Lubricants
RGT	Regiment
ROSS	Rule Oriented Simulation System
SA	Surface-to-Air (Missile)
SCORES	Scenario Oriented Recurring Evaluation System
SCUD	Surface-to-Surface Missile
SPL	Secure Processing Laboratory
S&S	Supply and Service
TGT	Target
TK	Tank
VAX	VAX 11/780 Computer

AD-A143 753 THE BATTLEFIELD ENVIRONMENT MODEL (BEM)(U) MITRE CORP 3/3
MCLEAN VA MITRE C3I DIV R S CONKER ET AL. SEP 83
MTR-83W00245 F19628-84-C-0001

UNCLASSIFIED F/G 15/7 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

REFERENCES

1. The MITRE Corporation, The MITRE Threat Event Generator - Force Movement, MTR-80W00266, Russell P. Bonasso and Emanuel P. Maimone, March 1981.

2. The MITRE Corporation, The MITRE Threat Event Generator (MTEG) - Force Communications, MTR 81W00295, R. P. Bonasso, December 1981.

3. The MITRE Corporation, ANALYST: An Expert System for Processing Sensor Returns, MTP-83W00002, R. P. Bonasso, 1983.

4. The MITRE Corporation, A Preliminary Evaluation of Object-Oriented Programming for Ground Combat Modeling, WP-83W00407, R. O. Nugent, September 1983.

5. The RAND Corporation, The ROSS Language Manual, N-1884-AF, David McArthur and Philip Klahr, September 1982.

6. The RAND Corporation, SWIRL: Simulating Warfare in the ROSS Language, N-1885-AF, Philip Klahr, David McArthur, Sanjai Narain, Eric Best, September 1982.

7. U. S. Army Combined Developments Activity, Standard Scenario for Combat Developments, Europe I, Sequence 2A (U), SECRET.

8. The MITRE Corporation, Threat Definition: Unit Locations and Target Arrarys for Soviet Artillery Battalions and Maneuver Battalions, WP-12724, R. R. Darron and D. T. Giles, Jr., 15 November 1977, SECRET.

9. The MITRE Corporation, Threat Definition: Combat Service Support Units, WP-12843, R. R. Darron and D. T. Giles, Jr., 10 February 1978, SECRET.

DISTRIBUTION LIST

INTERNAL

A-10 C. A. Zraket

D-11 A. J. Roberts

D-14 E. C. Brady

W-32 P. K. Groveston (3)

W-70 P. G. Freck
R. P. Granato
R. A. Joy
F. W. Niedenfuhr

W-72 C. W. Sanders

W-73 T. H. Nyman
V. Omeally

W-74 H. J. Antonisse
T. T. Bean
R. P. Bonasso
M. Gale
C. R. Holt
S. J. Laskowski
A. M. Lidy
R. O. Nugent (10)
T. F. Turner

W-76 R. G. Gados
E. J. Kirk
E. P. Maimone
W. E. Zeiner

W-93 R. S. Conker (3)
J. W. Benoit
J. R. Davidson (3)

W-70 Information Center

EXTERNAL

HQ, Department of the Army
SAUS-OR (Mr. Hollis,)
DAMO-ZD (Mr. Vandiver)
DAMA-ZD (Mr. Woodall)
DAMI-FRT (Mr. Beuch)
DACS-DMO (Ms. Langston)
ASA (IL&FM) (Mr. Rosenblum)
Washington D.C. 20310

Commander
U.S. Army Electronics Research and Development Command
2800 Powder Mill Road
Adelphi, MD 20783

Director Electronic Warfare Lab
PM-SOTAS
Fort Monmouth, NJ 07703

Director Signals Warfare Lab
PM-JTFS
Vint Hill Farms Station
Warrenton, VA 22186

Jet Propulsion Laboratories
Mr. S. Friesma
4800 Oak Grove
Pasadena, CA 91109

Institute for Defense Analysis
1801 North Beauregard Street
Alexandria, VA 22311

The Army Model Management Office
U.S. Army Combined Arms Center
ATTN: ATZL-CAN-DO
COL Kenneth E. Wiersema
Ft. Leavenworth, KS 66027 (2)

DISTRIBUTION LIST

(Continued)

EXTERNAL

Army Library
ATTN: ANR-AL-RS
(Army Studies)
Room 1A518
Pentagon
Washington D.C. 20310

Commander
Defense Technical Information Center
ATTN: DDA
Cameron Station
Alexandria, VA 22314 (2)

Commandant
U.S. Army Command and General Staff College
Ft. Leavenworth, KS 66027

Commandant
U.S. Army War College
Carlisle Barracks, PA 17013

HQ DARCOM
DRCBSI
5001 Eisenhower Avenue
Alexandria, VA 22333

Commander
U.S. Army Combined Arms Center
ATZL-CSC-I
ATZL-CAS-W
ATZL-TAC-LO
Ft. Leavenworth, KS 66027

Commander
U.S. Army Nuclear and Chemical Agency
ATTN: MONA-OPS
Ft. Belvoir, VA 22060

Commander
U.S. Army Infantry Center
ATTN: ATSH-CD-CSO-OR
Ft. Benning, GA 31905

Commander
U.S. Army Aviation Center
ATTN: ATZQ-D-CS
Ft. Rucker, AL 36362

Commander
U.S. Army Air Defense School
ATTN: ATSA-CDS-F
Ft. Bliss, TX 79916

Commander
U.S. Army Logistics Center
ATTN: ATCL-O
Ft. Lee, VA 23801

Commander
U.S. Army Field Artillery School
ATTN: ATSF-CA
Ft. Sill, OK 73503

Commander
U.S. Army Armor Center
ATTN: ATZK-CD-SD
Ft. Knox, KY 40121

Commander
U.S. Army Intelligence Center
ATTN: ATSI-CD-CS
(LTC Aikens, Dr. Verhey)
Ft. Huachuca, AZ 85613

Commander
Soldier Support Center
ATTN: ATSG-DCD-AD
Ft. Benjamin Harrison, IN 46216

DISTRIBUTION LIST

(Concluded)

EXTERNAL

Director, U.S. Army Concepts
Analysis Agency
ATTN: CSCA-AZ
CSCA-MC
8120 Woodmont Avenue
Bethesda, MD 20014

Director, U.S. Army Materiel Systems
Analysis Activity
ATTN: DRXSY-C (Mr. Myers)
DRXSY-GR (Mr. Clifford)
Aberdeen Proving Ground, MD 21005

Commander
Director, U.S. Army TRADOC Systems
Analysis Activity
ATTN: ATAA-D (Mr. Goode)
ATAA-TG (Mr. Carrillo)
ATAA-TC (Mr. Matheson)
ATAA-TC (Mr. Payan)
White Sands, NM 88002

Director, U.S. Army Research Institute
ATTN: PERI-SZ (Dr. Johnson)
5001 Eisenhower Avenue
Alexandria, VA 22333

Dr. Wilbur Payne
Director TRADOC Operations Research
Activity
White Sands, NM 88002

Deputy Commander, Combined Arms
Systems Analysis Activity
ATTN: ATOR-CAA-DC
ATOR-CAA-DR
Ft. Leavenworth, KS 66027

Commander
U.S. Army Intelligence and Threat
Analysis Center
ATTN: IAX-I
IAX-I-OR
Arlington Hall Station, VA 22022

Commander
U.S. Army Training and Doctrine
Command
ATTN: ATCG-S (Mr. Christman)
Ft. Monroe, VA 23651

Lawrence Livermore Laboratory
Attn: Stan Erickson, L-7
P.O. Box 808
Livermore, CA 94550

Ray Kirkwood
TRADOC Combined Arms Test Activity
Ft. Hood, Texas 76544

Major Kevin Dolan
JFAAD
Ft. Bliss, TX 79916

The Rand Corporation
Dr. Philip Klahr
1700 Main Street
Santa Monica, VA

RADC/IRDT
Andrew Kozak
Griffiss AFB, NY 13441

Dr. Northrop Fowler
RADC/COES
Griffiss AFB, NY 13441

END

FILMED

DTIC